МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙУНИВЕРСИТЕТ им. В. Г. Шухова»
(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения
вычислительнойтехники и автоматизированных
систем

Лабораторная работа № 13
По дисциплине
Основы программирования
По теме: «Множества»

Выполнил: ст. группы КБ – 231

Давыденко Кирилл Иванович

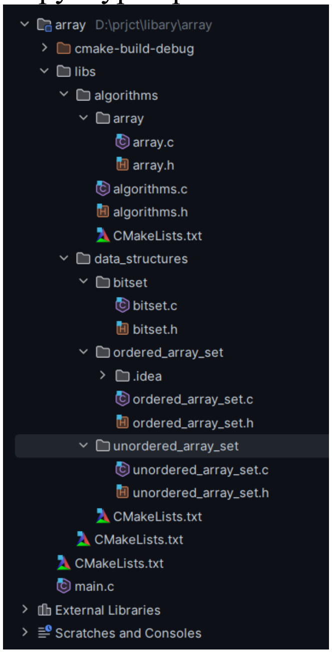
Проверили:
Черников Сергей Викторович
Новожен Никита Викторович

Белгород, 2023 г.

Цель работы: закрепление навыков работы со структурами, изучение простых способов представления множеств в памяти ЭВМ.

Структура отчёта:

Структура проекта:



```
array  D:\prjct\libary\array
  cmake-build-debug
  libs
    algorithms
      array
        array.c
        array.h
      algorithms.c
      algorithms.h
      CMakeLists.txt
    data_structures
      bitset
        bitset.c
        bitset.h
      ordered_array_set
        .idea
        ordered_array_set.c
        ordered_array_set.h
      unordered_array_set
        unordered_array_set.c
        unordered_array_set.h
      CMakeLists.txt
    CMakeLists.txt
  CMakeLists.txt
  main.c
External Libraries
Scratches and Consoles
```

CMakeList проекта:

```
cmake_minimum_required(VERSION 3.26)
project(array C)

set(CMAKE_C_STANDARD 11)

add_executable(array main.c
        libs/algorithms/array/array.c
        libs/algorithms/array/array.h
        libs/algorithms/algorithms.c
        libs/algorithms/algorithms.h)

add_subdirectory(libs)
target_link_libraries(array data_structures)
target_link_libraries(array algorithms)
```

CMakeList папки libs:

```
add_subdirectory(algorithms)
add_subdirectory(data_structures)
```

CMakeList папки algorithms:

```
add_library(algorithms
        algorithms.c
        array/array.c
)
```

CMakeList папки data_structures:

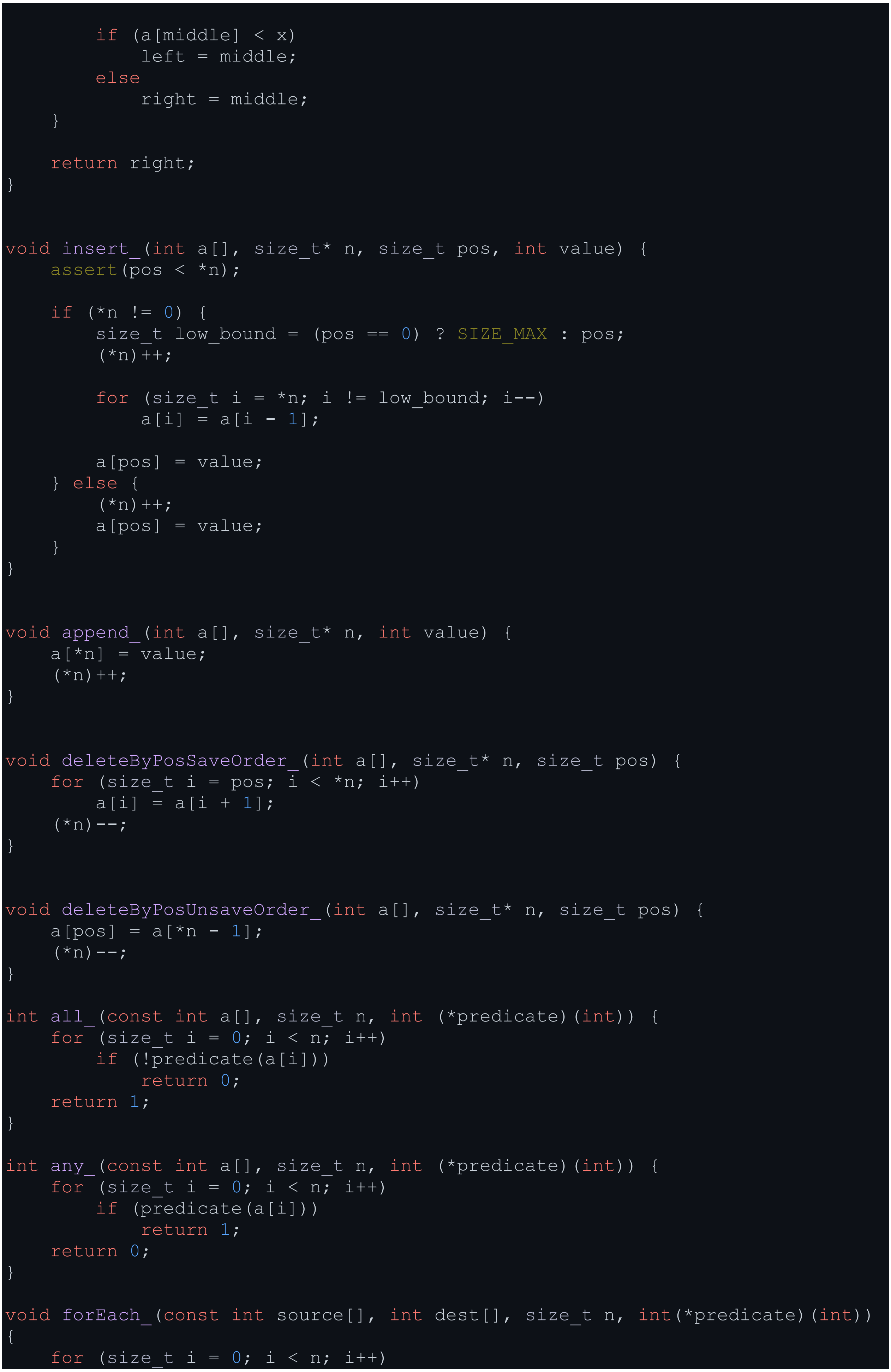
```
add_library(data_structures
        bitset/bitset.c
        bitset/bitset.h
        unordered_array_set/unordered_array_set.h
        unordered_array_set/unordered_array_set.c
        ordered_array_set/ordered_array_set.c
        ordered_array_set/ordered_array_set.h
)
```

Содержание array.h:

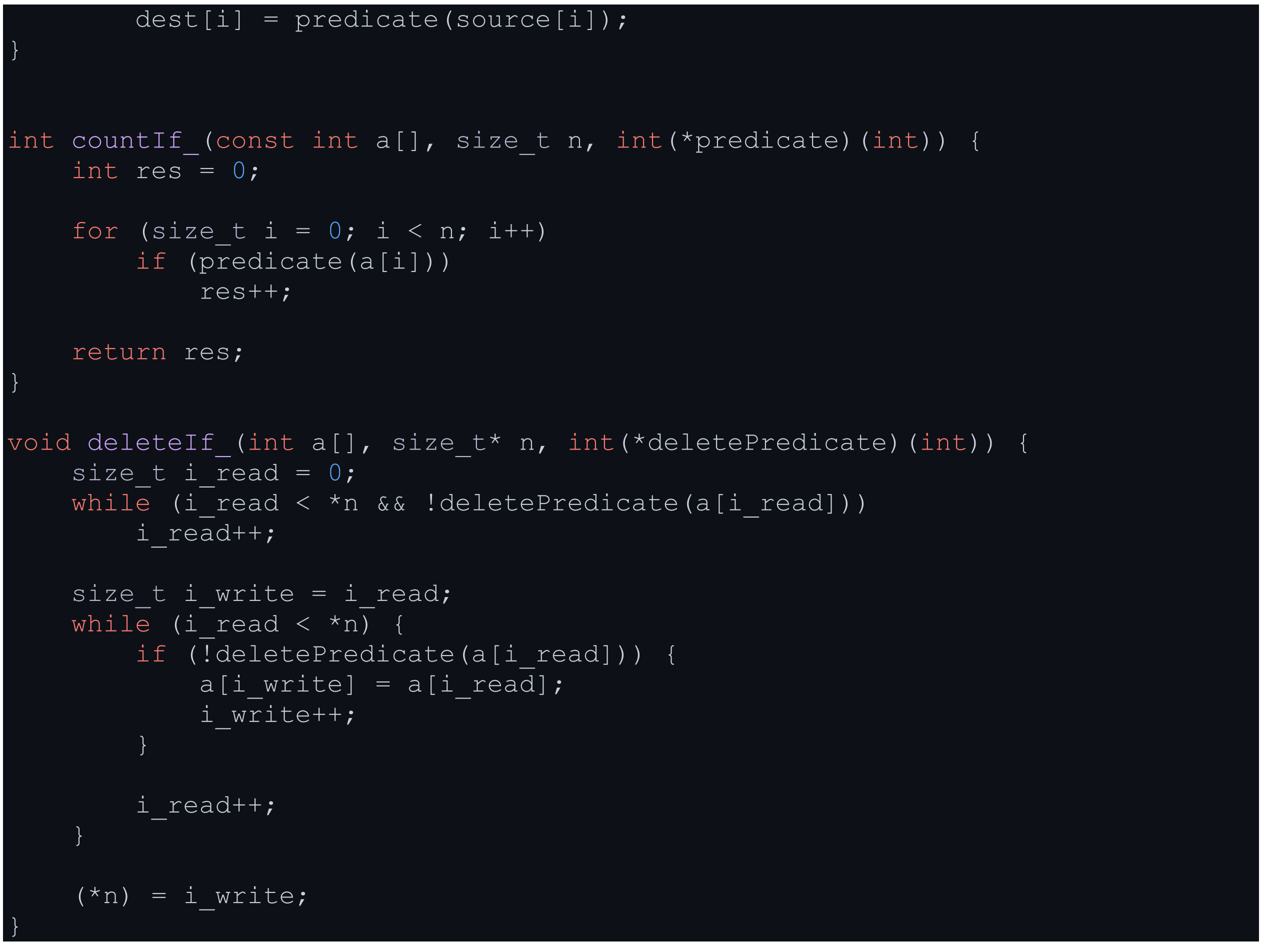
```
#ifndef ARRAY_ARRAY_H
#define ARRAY_ARRAY_H

#include <stddef.h>


// ввод массива a размера n
void inputArray_(int a[], size_t n);

// вывод массива a размера n
void outputArray_(const int a[], size_t n);

// возвращает индекс первого вхождения элемента x
// в массиве a размера n при наличии, иначе n
size_t linearSearch_(const int a[], const size_t n, int x);

// возвращает индекс первого вхождения элемента x
// в отсортированном массиве a размера n при наличии, иначе n
size_t binarySearch_(const int a[], const size_t n, int x);

// возвращает индекс первого элемента равного или большего x
// в отсортированном массиве a размера n
// при отсутствии возврощает n
size_t binarySearchMoreOrEqual_(const int a[], const size_t n, int x);

// вставка элемента со значением value
// в массив a размера n на позицию pos
void insert_(int a[], size_t* n, size_t pos, int value);

// вставка элемента со значением value
// в конец массива a размера n
void append_(int a[], size_t* n, int value);

// удаление из массива a размера n элемента на позиции pos
// с сохранением порядка оставшихся элементов
void deleteByPosSaveOrder_(int a[], size_t* n, size_t pos);

// удаление из массива a размера n элемента на позиции pos
// без сохранения порядка оставшихся элементов
// размер массива a уменьшается на единицу
void deleteByPosUnsaveOrder_(int a[], size_t* n, size_t pos);

// возвращает истина, если все элементы массива a размера n
// удовлетворяют функции-предикату predicate
// иначе ложь
int all_(const int a[], size_t n, int (*predicate)(int));

// возвращает истина, если один элемент массива a размера n
// удовлетворяют функции-предикату predicate
// иначе ложь
int any_(const int a[], size_t n, int (*predicate)(int));

// применяет фукнцию-предикат predicate ко всем элементам массива source
// сохраняет результат в массив dest размера n
void forEach_(const int source[], int dest[], size_t n,
int(*predicate)(int));

// возвращает количество элементов массива a размера n
// удовлетворяющих функции-предикату predicate
int countIf_(const int a[], size_t n, int(*predicate)(int));

// удаляет из массива a размера n все элементы, соответсвующие
// функции-предикату deletePredicate, записывает в n
// новый размер массива
void deleteIf_(int a[], size_t* n, int(*deletePredicate)(int));

#endif //ARRAY_ARRAY_H
```

Содержание array.c:

```
#include <stdio.h>
#include <limits.h>
#include <assert.h>
#include "array.h"


void inputArray_(int a[], const size_t n) {
    for (size_t i = 0; i < n; i++)
        scanf("%d", a + i);
}


void outputArray_(const int a[], const size_t n) {
    for (size_t i = 0; i < n; i++)
        printf("%d ", *(a + i));
    printf("\n");
}


size_t linearSearch_(const int a[], const size_t n, int x) {
    for (size_t i = 0; i < n; i++)
        if (a[i] == x)
            return i;
    return n;
}


size_t binarySearch_(const int a[], const size_t n, int x) {
    if (a[0] > x || n == 0)
        return n;

    size_t left = 0;
    size_t right = n - 1;

    while (left <= right) {
        size_t middle = left + (right - left) / 2;

        if (a[middle] < x)
            left = middle + 1;
        else if (a[middle] > x)
            right = middle - 1;
        else
            return middle;
    }

    return n;
}


size_t binarySearchMoreOrEqual_(const int a[], const size_t n, int x) {
    if (a[0] >= x)
        return 0;

    size_t left = 0;
    size_t right = n;

    while (right - left > 1) {
        size_t middle = left + (right - left) / 2;
```

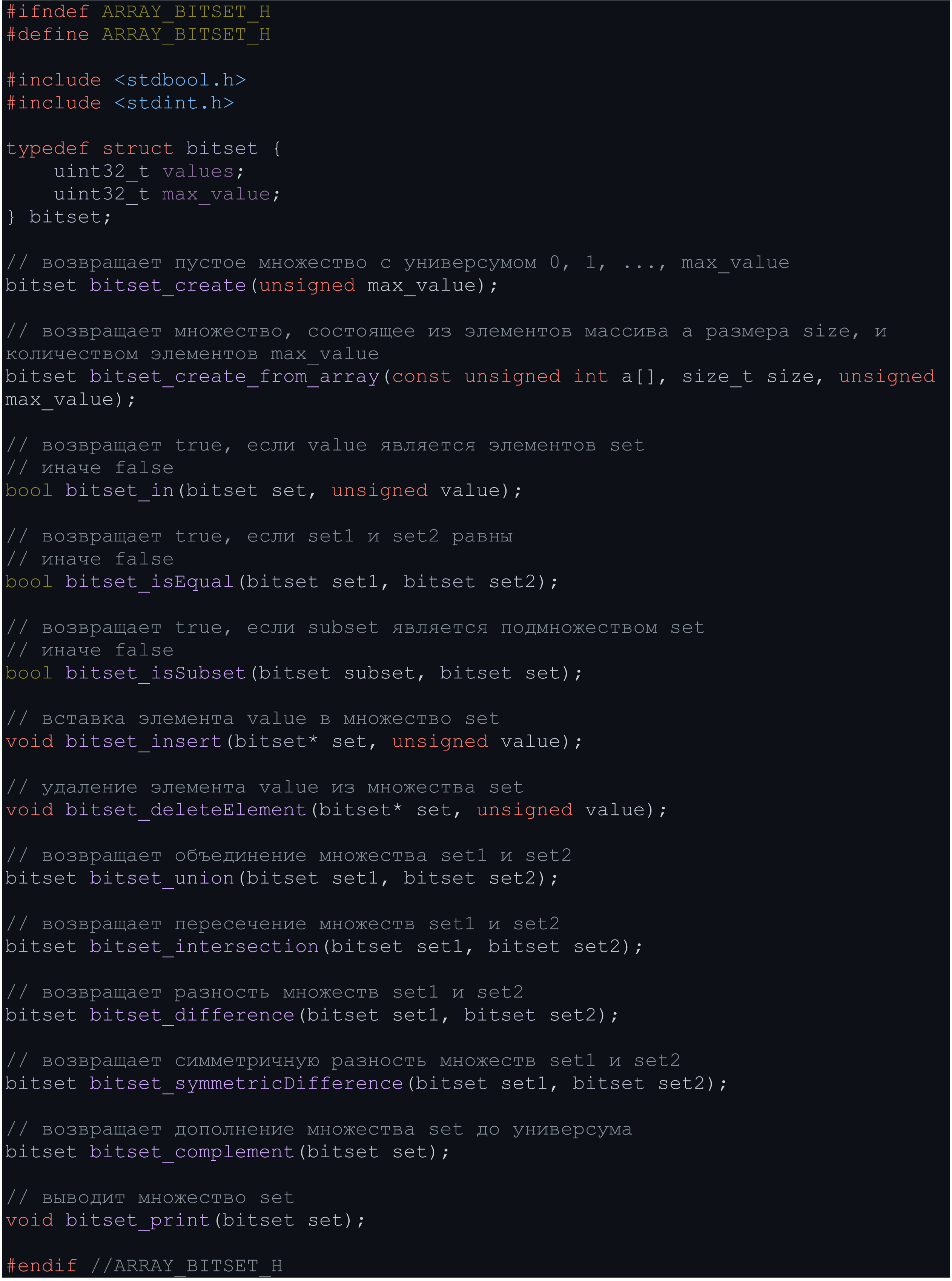


```
        if (a[middle] < x)
            left = middle;
        else
            right = middle;
    }

    return right;
}


void insert_(int a[], size_t* n, size_t pos, int value) {
    assert(pos < *n);

    if (*n != 0) {
        size_t low_bound = (pos == 0) ? SIZE_MAX : pos;
        (*n)++;

        for (size_t i = *n; i != low_bound; i--)
            a[i] = a[i - 1];

        a[pos] = value;
    } else {
        (*n)++;
        a[pos] = value;
    }
}


void append_(int a[], size_t* n, int value) {
    a[*n] = value;
    (*n)++;
}


void deleteByPosSaveOrder_(int a[], size_t* n, size_t pos) {
    for (size_t i = pos; i < *n; i++)
        a[i] = a[i + 1];
    (*n)--;
}


void deleteByPosUnsaveOrder_(int a[], size_t* n, size_t pos) {
    a[pos] = a[*n - 1];
    (*n)--;
}

int all_(const int a[], size_t n, int (*predicate)(int)) {
    for (size_t i = 0; i < n; i++)
        if (!predicate(a[i]))
            return 0;
    return 1;
}

int any_(const int a[], size_t n, int (*predicate)(int)) {
    for (size_t i = 0; i < n; i++)
        if (predicate(a[i]))
            return 1;
    return 0;
}

void forEach_(const int source[], int dest[], size_t n, int(*predicate)(int))
{
    for (size_t i = 0; i < n; i++)
```



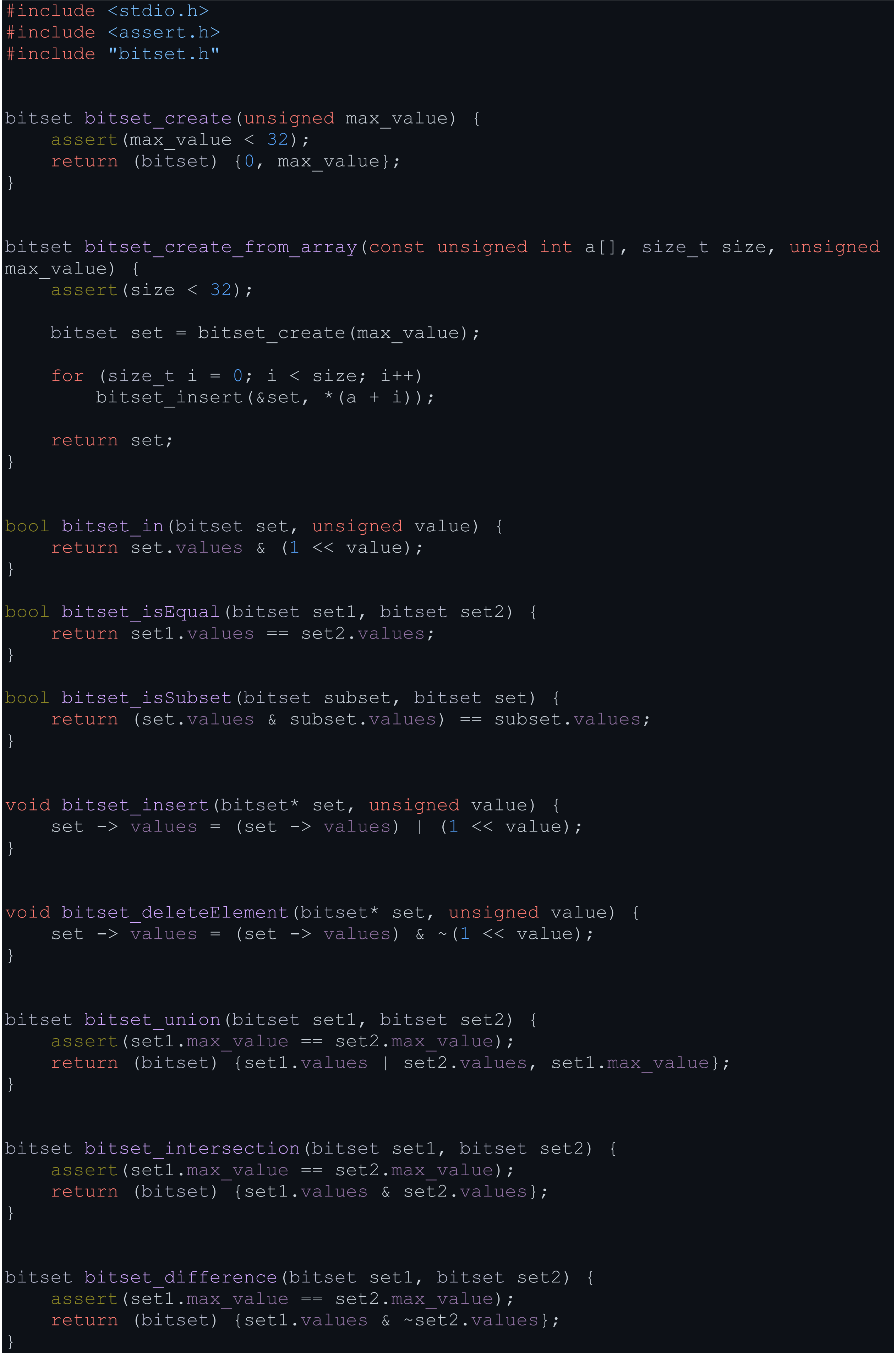
```c
        dest[i] = predicate(source[i]);
}


int countIf_(const int a[], size_t n, int(*predicate)(int)) {
    int res = 0;

    for (size_t i = 0; i < n; i++)
        if (predicate(a[i]))
            res++;

    return res;
}

void deleteIf_(int a[], size_t* n, int(*deletePredicate)(int)) {
    size_t i_read = 0;
    while (i_read < *n && !deletePredicate(a[i_read]))
        i_read++;

    size_t i_write = i_read;
    while (i_read < *n) {
        if (!deletePredicate(a[i_read])) {
            a[i_write] = a[i_read];
            i_write++;
        }

        i_read++;
    }

    (*n) = i_write;
}
```

Содержание bitset.h:

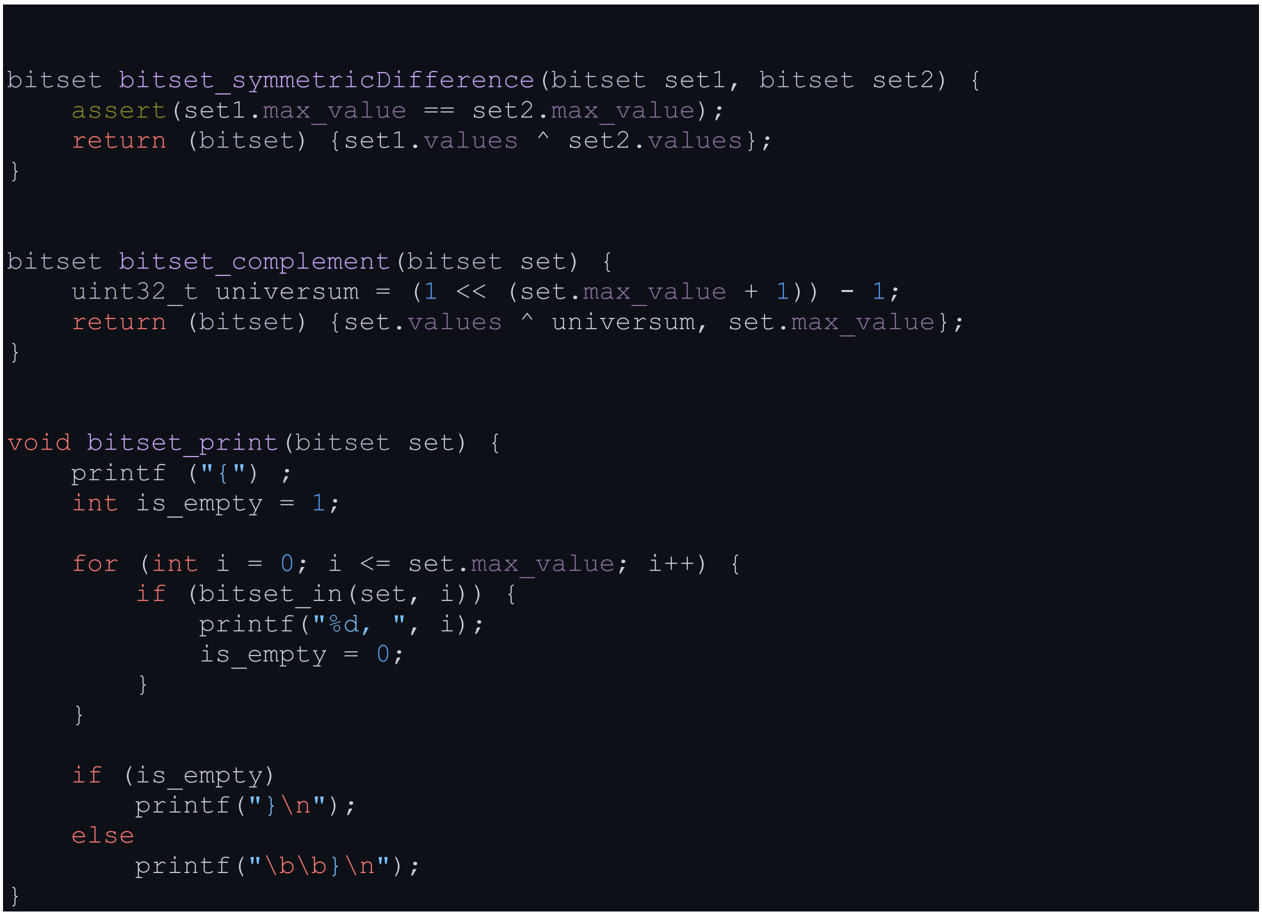


```c
#ifndef ARRAY_BITSET_H
#define ARRAY_BITSET_H

#include <stdbool.h>
#include <stdint.h>

typedef struct bitset {
    uint32_t values;
    uint32_t max_value;
} bitset;

// возвращает пустое множество с универсумом 0, 1, ..., max_value
bitset bitset_create(unsigned max_value);

// возвращает множество, состоящее из элементов массива a размера size, и
количеством элементов max_value
bitset bitset_create_from_array(const unsigned int a[], size_t size, unsigned
max_value);

// возвращает true, если value является элементов set
// иначе false
bool bitset_in(bitset set, unsigned value);

// возвращает true, если set1 и set2 равны
// иначе false
bool bitset_isEqual(bitset set1, bitset set2);

// возвращает true, если subset является подмножеством set
// иначе false
bool bitset_isSubset(bitset subset, bitset set);

// вставка элемента value в множество set
void bitset_insert(bitset* set, unsigned value);

// удаление элемента value из множества set
void bitset_deleteElement(bitset* set, unsigned value);

// возвращает объединение множества set1 и set2
bitset bitset_union(bitset set1, bitset set2);

// возвращает пересечение множеств set1 и set2
bitset bitset_intersection(bitset set1, bitset set2);

// возвращает разность множеств set1 и set2
bitset bitset_difference(bitset set1, bitset set2);

// возвращает симметричную разность множеств set1 и set2
bitset bitset_symmetricDifference(bitset set1, bitset set2);

// возвращает дополнение множества set до универсума
bitset bitset_complement(bitset set);

// выводит множество set
void bitset_print(bitset set);

#endif //ARRAY_BITSET_H
```

Содержание bitset.c:



```c
#include <stdio.h>
#include <assert.h>
#include "bitset.h"


bitset bitset_create(unsigned max_value) {
    assert(max_value < 32);
    return (bitset) {0, max_value};
}


bitset bitset_create_from_array(const unsigned int a[], size_t size, unsigned
max_value) {
    assert(size < 32);

    bitset set = bitset_create(max_value);

    for (size_t i = 0; i < size; i++)
        bitset_insert(&set, *(a + i));

    return set;
}


bool bitset_in(bitset set, unsigned value) {
    return set.values & (1 << value);
}

bool bitset_isEqual(bitset set1, bitset set2) {
    return set1.values == set2.values;
}

bool bitset_isSubset(bitset subset, bitset set) {
    return (set.values & subset.values) == subset.values;
}


void bitset_insert(bitset* set, unsigned value) {
    set -> values = (set -> values) | (1 << value);
}


void bitset_deleteElement(bitset* set, unsigned value) {
    set -> values = (set -> values) & ~(1 << value);
}


bitset bitset_union(bitset set1, bitset set2) {
    assert(set1.max_value == set2.max_value);
    return (bitset) {set1.values | set2.values, set1.max_value};
}


bitset bitset_intersection(bitset set1, bitset set2) {
    assert(set1.max_value == set2.max_value);
    return (bitset) {set1.values & set2.values};
}


bitset bitset_difference(bitset set1, bitset set2) {
    assert(set1.max_value == set2.max_value);
    return (bitset) {set1.values & ~set2.values};
}
```



```c
bitset bitset_symmetricDifference(bitset set1, bitset set2) {
    assert(set1.max_value == set2.max_value);
    return (bitset) {set1.values ^ set2.values};
}


bitset bitset_complement(bitset set) {
    uint32_t universum = (1 << (set.max_value + 1)) - 1;
    return (bitset) {set.values ^ universum, set.max_value};
}


void bitset_print(bitset set) {
    printf ("{") ;
    int is_empty = 1;

    for (int i = 0; i <= set.max_value; i++) {
        if (bitset_in(set, i)) {
            printf("%d, ", i);
            is_empty = 0;
        }
    }

    if (is_empty)
        printf("}\n");
    else
        printf("\b\b}\n");
}
```

Содержание ordered_array_set.h:

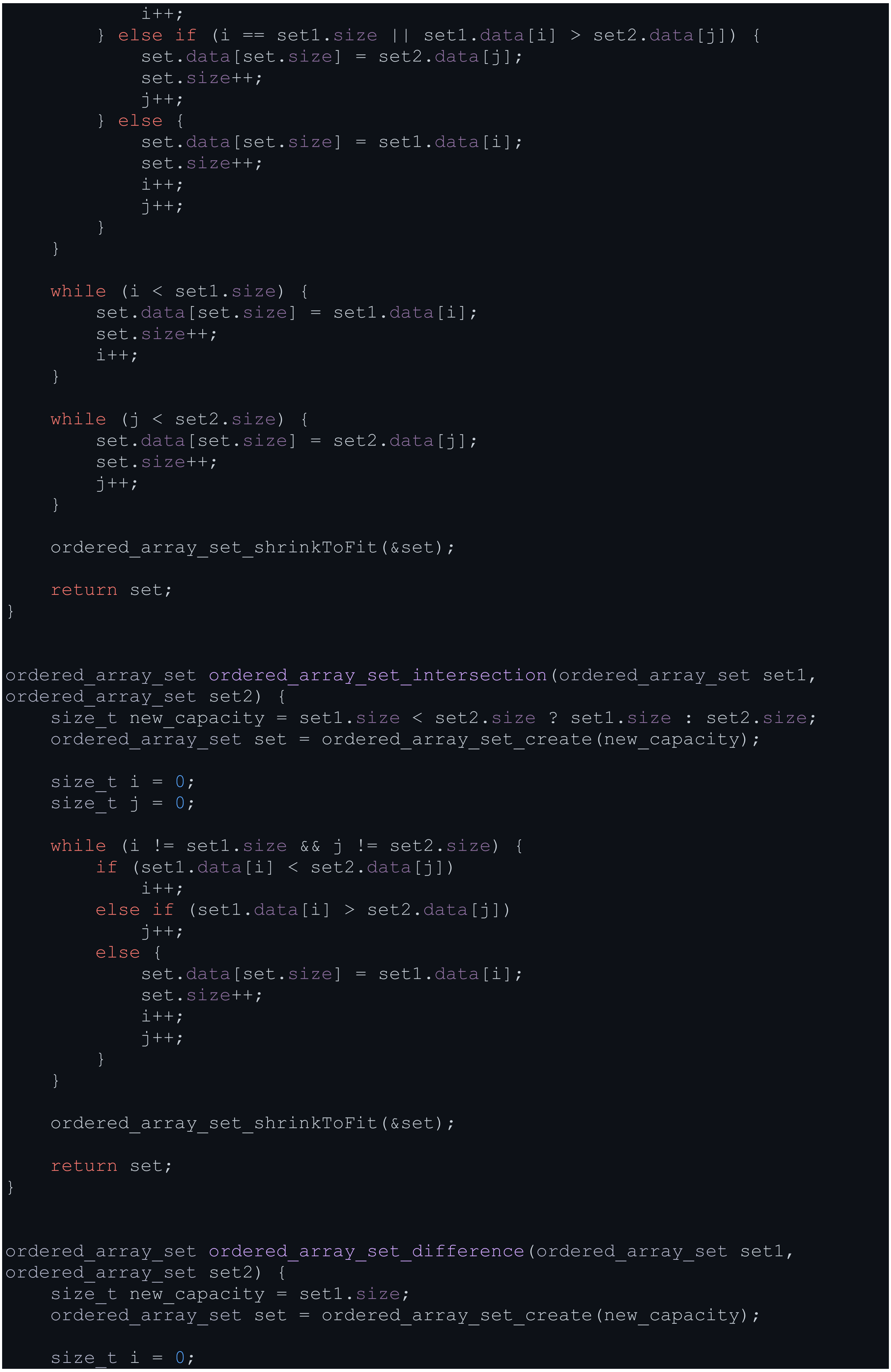
```
#ifndef ARRAY_ORDERED_ARRAY_SET_H
#define ARRAY_ORDERED_ARRAY_SET_H


#include <stdint.h>
#include <assert.h>
#include <memory.h>
#include <stdio.h>
#include <stdbool.h>
#include "../../algorithms/array/array.h"


typedef struct ordered_array_set {
    int* data;
    size_t size;
    size_t capacity;
} ordered_array_set;


// возвращает пусткое множество для capacity элементов
ordered_array_set ordered_array_set_create(size_t capacity);

// возвращает множество, состоящее из элементов массива a размера size
ordered_array_set ordered_array_set_create_from_array(const int a[], size_t
size);

// возвращает позицию элемента в множестве,
// если значение value имеется в множестве set, иначе n
size_t ordered_array_set_in(ordered_array_set* set, int value);

// возвращает true, если subset является подмножеством set.
// инчае false
bool ordered_array_set_isSubset(ordered_array_set subset, ordered_array_set
set);

// возвращает true, если элементы множеств set1 и set2 равны
// иначе false
bool ordered_array_set_isEqual(ordered_array_set set1, ordered_array_set
set2);

// возбуждает исплючение, если в множестве по адресу set
// нельзя вставить элемент
void ordered_array_set_isAbleAppend(ordered_array_set *set);

// добавляет элемент value в множество set
void ordered_array_set_insert(ordered_array_set* set, int value);

// удалить элемент value из множества set
void ordered_array_set_deleteElement(ordered_array_set* set, int value);

// возвращает объединение множеств set1 и set2
ordered_array_set ordered_array_set_union(ordered_array_set set1,
ordered_array_set set2);

// возвращает пересечение множеств set1 и set2
ordered_array_set ordered_array_set_intersection(ordered_array_set set1,
ordered_array_set set2);

// возвращает разность множеств set1 и set2
ordered_array_set ordered_array_set_difference(ordered_array_set set1,
ordered_array_set set2);

// возвращает дополнение множества set до универсума universumSet
```

```
ordered_array_set ordered_array_set_complement(ordered_array_set set,
ordered_array_set universumSet);

// возвращает симметричную разность множеств set1 и set2
ordered_array_set ordered_array_set_symmetricDifference(ordered_array_set
set1, ordered_array_set set2);

// вывод множества set
void ordered_array_set_print(ordered_array_set set);

// освобождает память, занимаемую множеством set
void ordered_array_set_delete(ordered_array_set* set);


#endif //ARRAY_ORDERED_ARRAY_SET_H
```

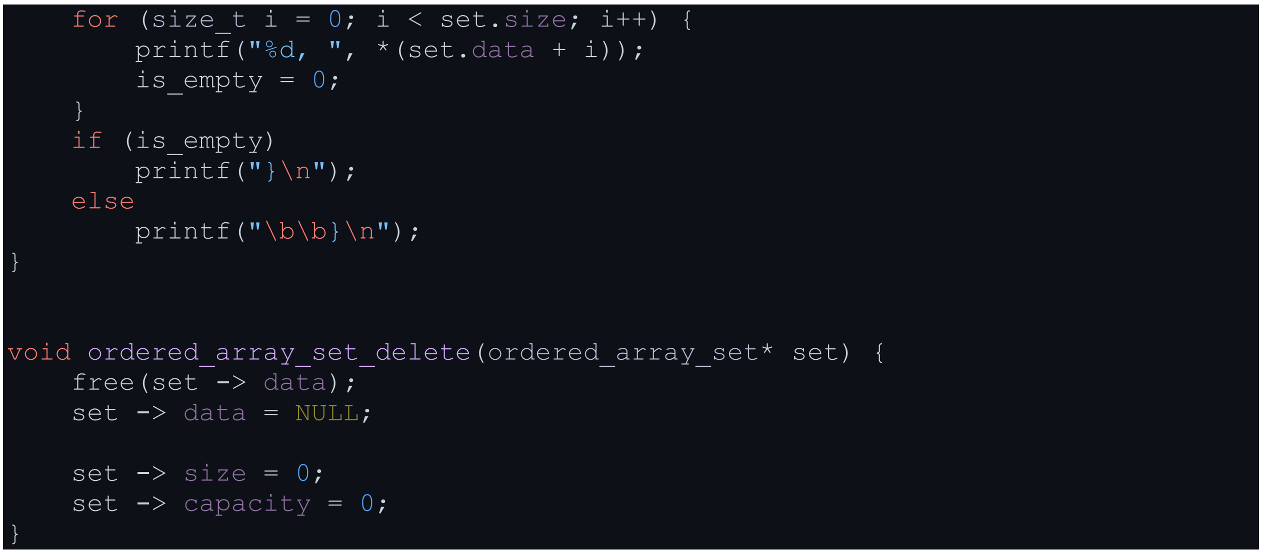
Содержание ordered_array_set.c:

```
#include <stdio.h>
#include <assert.h>
#include <malloc.h>
#include <stdlib.h>
#include <memory.h>
#include "../../algorithms/array/array.h"
#include "../../data_structures/ordered_array_set/ordered_array_set.h"


static int compare_ints(const void *a, const void *b) {
    return *(int *) a - *(int *) b;
}

ordered_array_set ordered_array_set_create(size_t capacity) {
    return (ordered_array_set) {malloc(sizeof(int) * capacity), 0, capacity};
}


void ordered_array_set_isAbleAppend(ordered_array_set *set) {
    assert(set->size < set->capacity);
}


size_t ordered_array_set_in(ordered_array_set *set, int value) {
    return binarySearch_(set->data, set->size, value);
}


void ordered_array_set_insert(ordered_array_set *set, int value) {
    size_t index = ordered_array_set_in(set, value);

    if (index == set->size) {
        ordered_array_set_isAbleAppend(set);

        size_t i;
        for (i = set->size; (i > 0 && set->data[i - 1] > value); i--)
            set->data[i] = set->data[i - 1];

        set->data[i] = value;
        set->size++;
    }
}


bool ordered_array_set_isEqual(ordered_array_set set1, ordered_array_set
set2) {
    if (set1.size != set2.size)
```

```
        return 0;

    return memcmp(set1.data, set2.data, sizeof(int) * set1.size) == 0;
}


void ordered_array_set_shrinkToFit(ordered_array_set *a) {
    if (a->size != a->capacity) {
        a->data = (int *) realloc(a->data, sizeof(int) * a->size);
        a->capacity = a->size;
    }
}


ordered_array_set ordered_array_set_create_from_array(const int *a, size_t
size) {
    ordered_array_set set = ordered_array_set_create(size);
    for (size_t i = 0; i < size; i++)
        ordered_array_set_insert(&set, *(a + i));
    ordered_array_set_shrinkToFit(&set);

    return set;
}


bool ordered_array_set_isSubset(ordered_array_set subset, ordered_array_set
set) {
    for (size_t i = 0; i < subset.size; i++) {
        bool found = false;

        for (size_t j = 0; j < set.size; j++)
            if (subset.data[i] == set.data[j]) {
                found = true;
                break;
            }

        if (!found)
            return false;
    }

    return true;
}


void ordered_array_set_deleteElement(ordered_array_set *set, int value) {
    size_t index = ordered_array_set_in(set, value);

    if (index != set->size)
        deleteByPosSaveOrder_(set->data, &set->size, index);
}


ordered_array_set ordered_array_set_union(ordered_array_set set1,
ordered_array_set set2) {
    size_t new_capacity = set1.size + set2.size;
    ordered_array_set set = ordered_array_set_create(new_capacity);


    size_t i = 0;
    size_t j = 0;
    while (i < set1.size && j < set2.size) {
        if (j == set2.size || set1.data[i] < set2.data[j]) {
            set.data[set.size] = set1.data[i];
            set.size++;
```

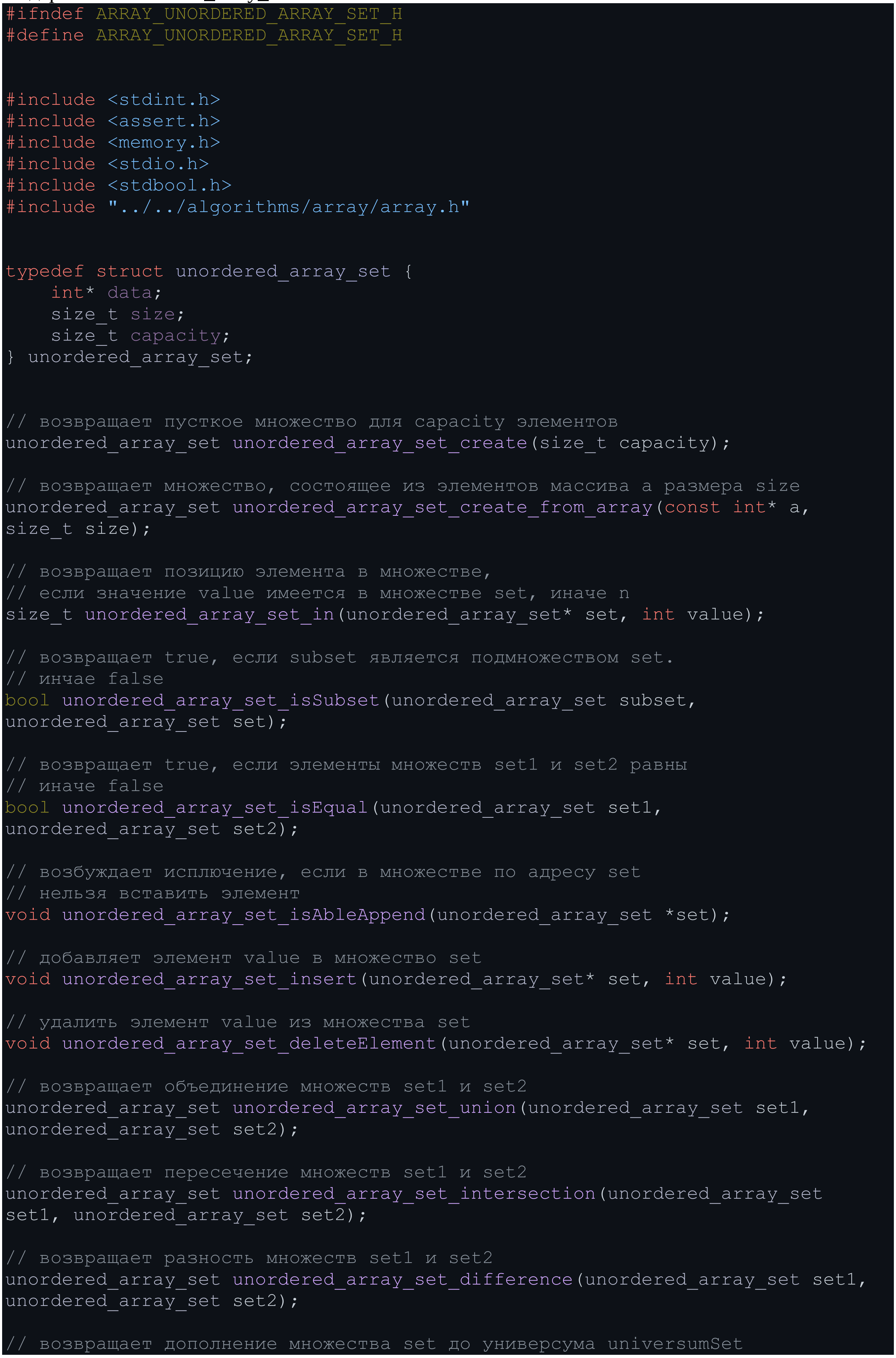


```c
            i++;
        } else if (i == set1.size || set1.data[i] > set2.data[j]) {
            set.data[set.size] = set2.data[j];
            set.size++;
            j++;
        } else {
            set.data[set.size] = set1.data[i];
            set.size++;
            i++;
            j++;
        }
    }

    while (i < set1.size) {
        set.data[set.size] = set1.data[i];
        set.size++;
        i++;
    }

    while (j < set2.size) {
        set.data[set.size] = set2.data[j];
        set.size++;
        j++;
    }

    ordered_array_set_shrinkToFit(&set);

    return set;
}


ordered_array_set ordered_array_set_intersection(ordered_array_set set1,
ordered_array_set set2) {
    size_t new_capacity = set1.size < set2.size ? set1.size : set2.size;
    ordered_array_set set = ordered_array_set_create(new_capacity);

    size_t i = 0;
    size_t j = 0;

    while (i != set1.size && j != set2.size) {
        if (set1.data[i] < set2.data[j])
            i++;
        else if (set1.data[i] > set2.data[j])
            j++;
        else {
            set.data[set.size] = set1.data[i];
            set.size++;
            i++;
            j++;
        }
    }

    ordered_array_set_shrinkToFit(&set);

    return set;
}


ordered_array_set ordered_array_set_difference(ordered_array_set set1,
ordered_array_set set2) {
    size_t new_capacity = set1.size;
    ordered_array_set set = ordered_array_set_create(new_capacity);

    size_t i = 0;
    size_t j = 0;

    while (i < set1.size) {
        if (j == set2.size || set1.data[i] < set2.data[j]) {
            set.data[set.size] = set1.data[i];
            set.size++;
            i++;
        } else if (set1.data[i] > set2.data[j])
            j++;
        else
            i++;
    }

    ordered_array_set_shrinkToFit(&set);

    return set;
}


ordered_array_set ordered_array_set_complement(ordered_array_set set,
ordered_array_set universumSet) {
    size_t new_capacity = universumSet.size;
    ordered_array_set new_set = ordered_array_set_create(new_capacity);

    size_t i = 0, j = 0;
    while (i < universumSet.size) {
        if (j < set.size && universumSet.data[i] == set.data[j]) {
            i++;
            j++;
        } else {
            new_set.data[new_set.size] = universumSet.data[i];
            new_set.size++;
            i++;
        }
    }

    ordered_array_set_shrinkToFit(&new_set);

    assert(ordered_array_set_isSubset(new_set, universumSet));

    return new_set;
}


ordered_array_set ordered_array_set_symmetricDifference(ordered_array_set
set1, ordered_array_set set2) {
    ordered_array_set universum = ordered_array_set_union(set1, set2);
    ordered_array_set intersection = ordered_array_set_intersection(set1,
set2);

    ordered_array_set symmetric = ordered_array_set_complement(intersection,
universum);

    ordered_array_set_delete(&intersection);
    ordered_array_set_delete(&universum);

    return symmetric;
}


void ordered_array_set_print(ordered_array_set set) {
    printf("{");
    int is_empty = 1;
```



```
    for (size_t i = 0; i < set.size; i++) {
        printf("%d, ", *(set.data + i));
        is_empty = 0;
    }
    if (is_empty)
        printf("}\n");
    else
        printf("\b\b}\n");
}


void ordered_array_set_delete(ordered_array_set* set) {
    free(set -> data);
    set -> data = NULL;

    set -> size = 0;
    set -> capacity = 0;
}
```

Содержание unordered_array_set.h:

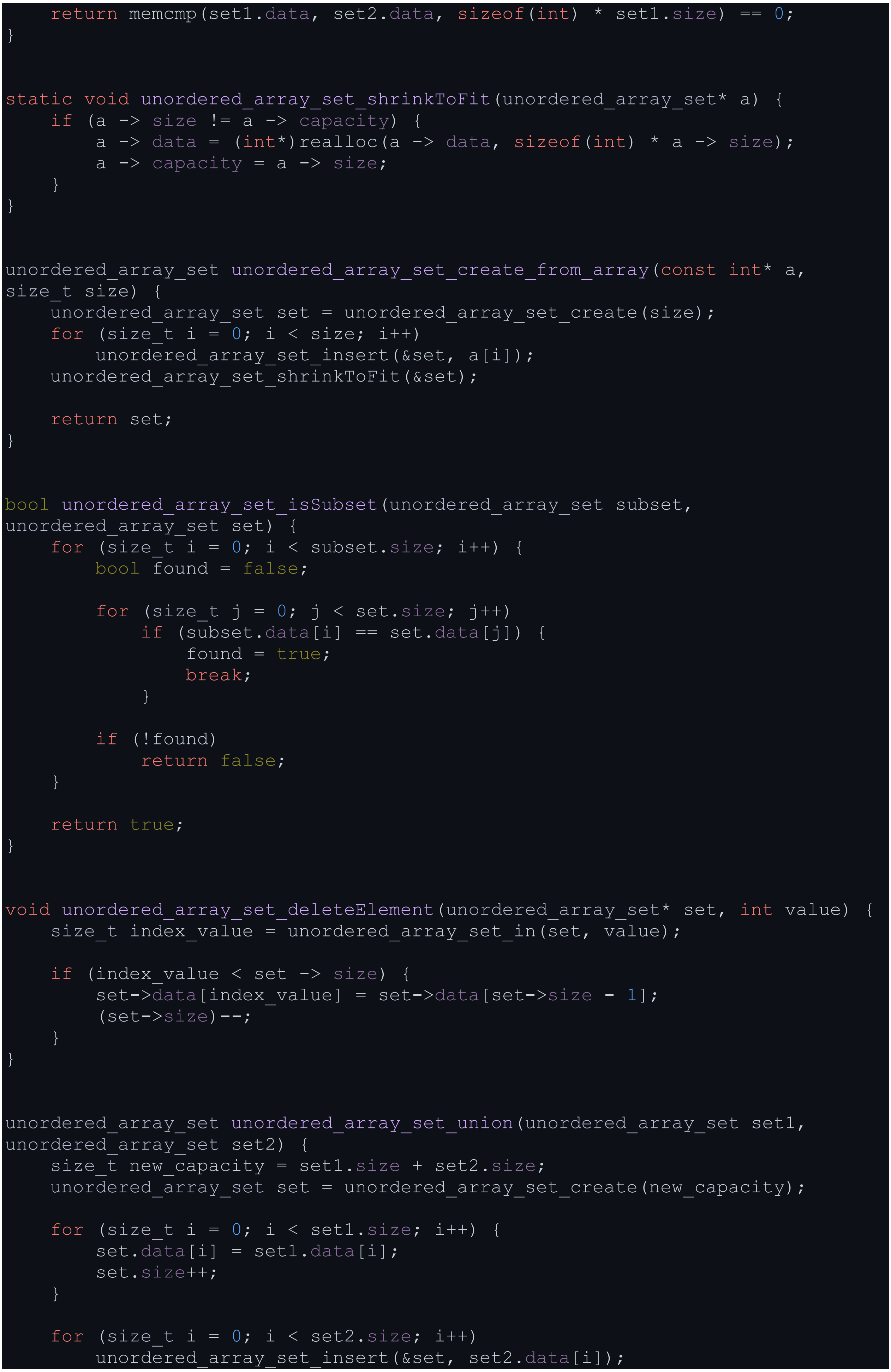


```
#ifndef ARRAY_UNORDERED_ARRAY_SET_H
#define ARRAY_UNORDERED_ARRAY_SET_H


#include <stdint.h>
#include <assert.h>
#include <memory.h>
#include <stdio.h>
#include <stdbool.h>
#include "../../algorithms/array/array.h"


typedef struct unordered_array_set {
    int* data;
    size_t size;
    size_t capacity;
} unordered_array_set;


// возвращает пусткое множество для capacity элементов
unordered_array_set unordered_array_set_create(size_t capacity);

// возвращает множество, состоящее из элементов массива a размера size
unordered_array_set unordered_array_set_create_from_array(const int* a,
size_t size);

// возвращает позицию элемента в множестве,
// если значение value имеется в множестве set, иначе n
size_t unordered_array_set_in(unordered_array_set* set, int value);

// возвращает true, если subset является подмножеством set.
// инчае false
bool unordered_array_set_isSubset(unordered_array_set subset,
unordered_array_set set);

// возвращает true, если элементы множеств set1 и set2 равны
// иначе false
bool unordered_array_set_isEqual(unordered_array_set set1,
unordered_array_set set2);

// возбуждает исплючение, если в множестве по адресу set
// нельзя вставить элемент
void unordered_array_set_isAbleAppend(unordered_array_set *set);

// добавляет элемент value в множество set
void unordered_array_set_insert(unordered_array_set* set, int value);

// удалить элемент value из множества set
void unordered_array_set_deleteElement(unordered_array_set* set, int value);

// возвращает объединение множеств set1 и set2
unordered_array_set unordered_array_set_union(unordered_array_set set1,
unordered_array_set set2);

// возвращает пересечение множеств set1 и set2
unordered_array_set unordered_array_set_intersection(unordered_array_set
set1, unordered_array_set set2);

// возвращает разность множеств set1 и set2
unordered_array_set unordered_array_set_difference(unordered_array_set set1,
unordered_array_set set2);

// возвращает дополнение множества set до универсума universumSet
```

```
unordered_array_set unordered_array_set_complement(unordered_array_set set,
unordered_array_set universumSet);

// возвращает симметричную разность множеств set1 и set2
unordered_array_set
unordered_array_set_symmetricDifference(unordered_array_set set1,
unordered_array_set set2);

// вывод множества set
void unordered_array_set_print(unordered_array_set set);

// освобождает память, занимаемую множеством set
void unordered_array_set_delete(unordered_array_set* set);


#endif //ARRAY_UNORDERED_ARRAY_SET_H
```

Содержание unordered_array_set.c:

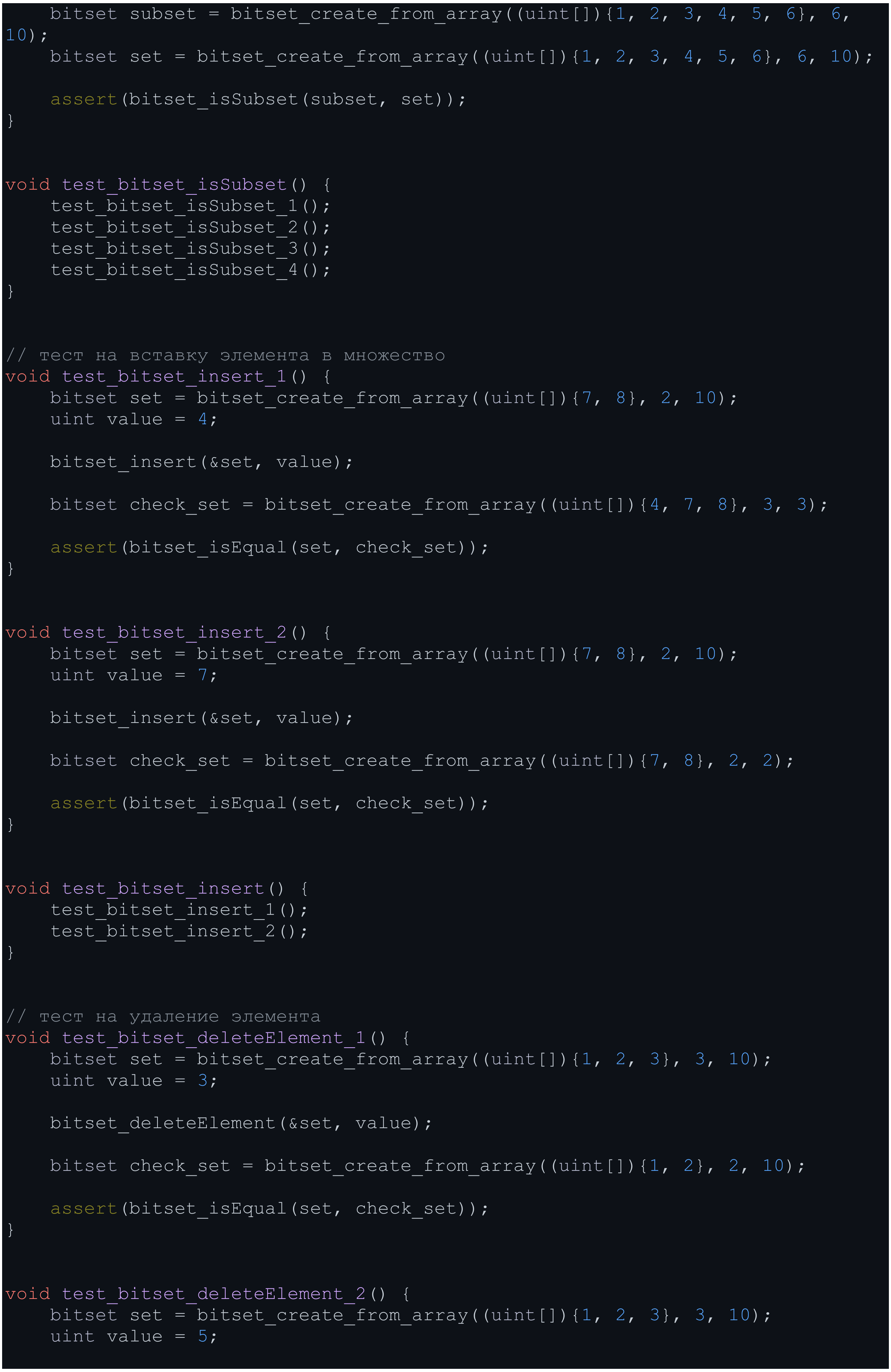
```
#include <stdio.h>
#include <assert.h>
#include <malloc.h>
#include <stdlib.h>
#include <memory.h>
#include "../../algorithms/array/array.h"
#include "../../data_structures/unordered_array_set/unordered_array_set.h"


static int compare_ints(const void* a, const void* b) {
    return *(int* ) a - *(int* ) b;
}


unordered_array_set unordered_array_set_create(size_t capacity) {
    return (unordered_array_set) {malloc(sizeof(int) * capacity), 0,
capacity};
}


void unordered_array_set_isAbleAppend(unordered_array_set *set) {
    assert(set -> size < set -> capacity);
}


size_t unordered_array_set_in(unordered_array_set* set, int value) {
    return linearSearch_(set -> data, set -> size, value);
}


void unordered_array_set_insert(unordered_array_set* set, int value) {
    if (unordered_array_set_in(set, value) == set -> size) {
        unordered_array_set_isAbleAppend(set);
        append_(set -> data, &set -> size, value);
    }
}


bool unordered_array_set_isEqual(unordered_array_set set1,
unordered_array_set set2) {
    if (set1.size != set2.size)
        return 0;

    qsort(set1.data, set1.size, sizeof(int), compare_ints);
    qsort(set2.data, set2.size, sizeof(int), compare_ints);
```



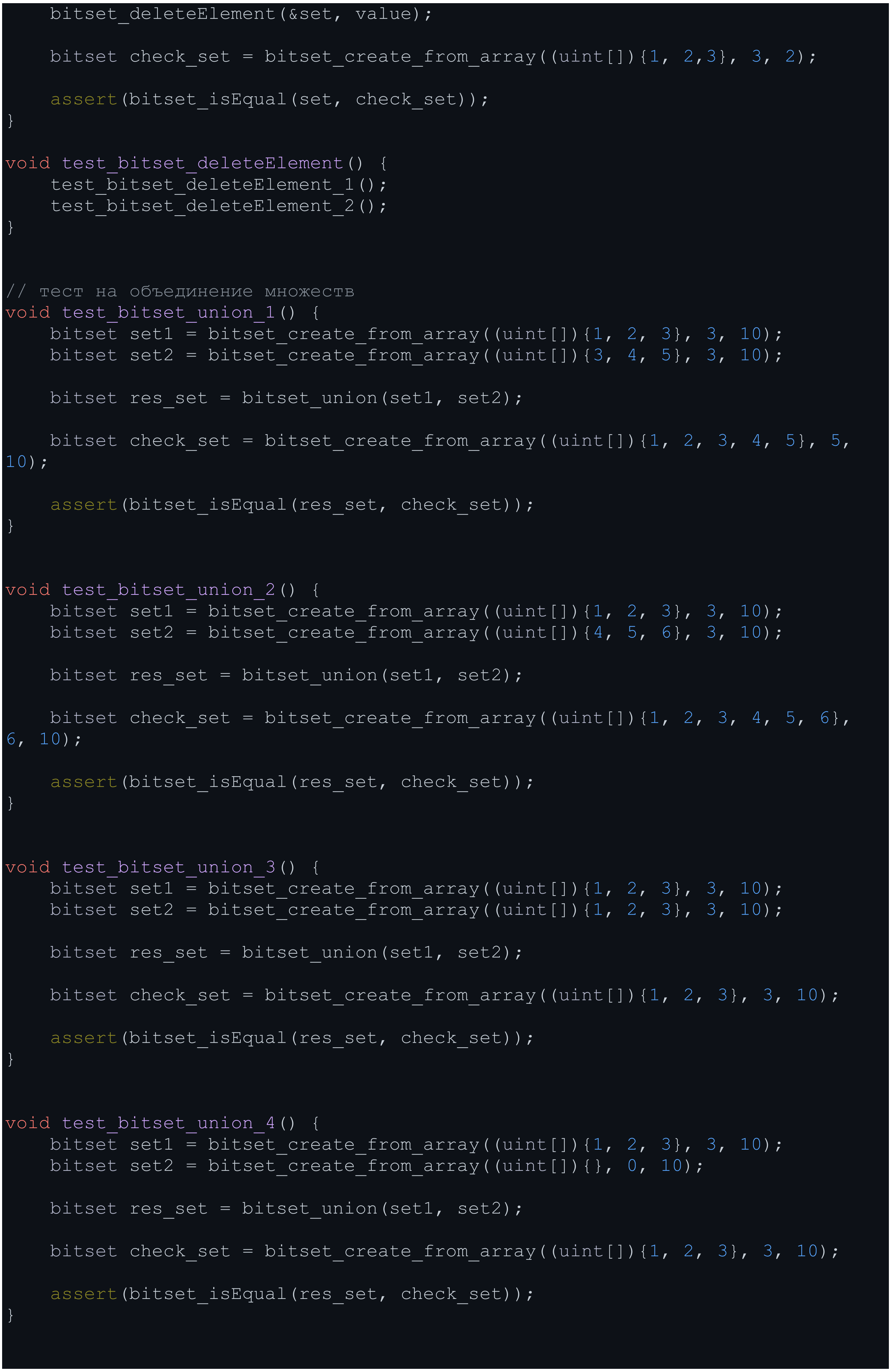
```
    return memcmp(set1.data, set2.data, sizeof(int) * set1.size) == 0;
}


static void unordered_array_set_shrinkToFit(unordered_array_set* a) {
    if (a -> size != a -> capacity) {
        a -> data = (int*)realloc(a -> data, sizeof(int) * a -> size);
        a -> capacity = a -> size;
    }
}


unordered_array_set unordered_array_set_create_from_array(const int* a,
size_t size) {
    unordered_array_set set = unordered_array_set_create(size);
    for (size_t i = 0; i < size; i++)
        unordered_array_set_insert(&set, a[i]);
    unordered_array_set_shrinkToFit(&set);

    return set;
}


bool unordered_array_set_isSubset(unordered_array_set subset,
unordered_array_set set) {
    for (size_t i = 0; i < subset.size; i++) {
        bool found = false;

        for (size_t j = 0; j < set.size; j++)
            if (subset.data[i] == set.data[j]) {
                found = true;
                break;
            }

        if (!found)
            return false;
    }

    return true;
}


void unordered_array_set_deleteElement(unordered_array_set* set, int value) {
    size_t index_value = unordered_array_set_in(set, value);

    if (index_value < set -> size) {
        set->data[index_value] = set->data[set->size - 1];
        (set->size)--;
    }
}


unordered_array_set unordered_array_set_union(unordered_array_set set1,
unordered_array_set set2) {
    size_t new_capacity = set1.size + set2.size;
    unordered_array_set set = unordered_array_set_create(new_capacity);

    for (size_t i = 0; i < set1.size; i++) {
        set.data[i] = set1.data[i];
        set.size++;
    }

    for (size_t i = 0; i < set2.size; i++)
        unordered_array_set_insert(&set, set2.data[i]);
    unordered_array_set_shrinkToFit(&set);

    return set;
}


unordered_array_set unordered_array_set_intersection(unordered_array_set
set1, unordered_array_set set2) {
    size_t new_capacity = set1.size < set2.size ? set1.size : set2.size;
    unordered_array_set set = unordered_array_set_create(new_capacity);

    for (size_t i = 0; i < set1.size; i++)
        if (unordered_array_set_in(&set2, set1.data[i]) != set2.size)
            unordered_array_set_insert(&set, set1.data[i]);

    return set;
}


unordered_array_set unordered_array_set_difference(unordered_array_set set1,
unordered_array_set set2) {
    size_t new_capacity = set1.size;
    unordered_array_set set = unordered_array_set_create(new_capacity);

    for (size_t i = 0; i < set1.size; i++)
        if (unordered_array_set_in(&set2, set1.data[i]) == set2.size)
            unordered_array_set_insert(&set, set1.data[i]);

    return set;
}


unordered_array_set unordered_array_set_complement(unordered_array_set set,
unordered_array_set universumSet) {
    size_t new_capacity = universumSet.size;
    unordered_array_set new_set = unordered_array_set_create(new_capacity);

    for (size_t i = 0; i < universumSet.size; i++)
        if (unordered_array_set_in(&set, universumSet.data[i]) == set.size)
            unordered_array_set_insert(&new_set, universumSet.data[i]);

    assert(unordered_array_set_isSubset(new_set, universumSet));

    return new_set;
}


unordered_array_set
unordered_array_set_symmetricDifference(unordered_array_set set1,
unordered_array_set set2) {
    unordered_array_set universum = unordered_array_set_union(set1, set2);
    unordered_array_set intersection = unordered_array_set_intersection(set1,
set2);

    unordered_array_set symmetric =
unordered_array_set_complement(intersection, universum);

    unordered_array_set_delete(&intersection);
    unordered_array_set_delete(&universum);

    return symmetric;
}
```

Тест работоспособности библиотек:

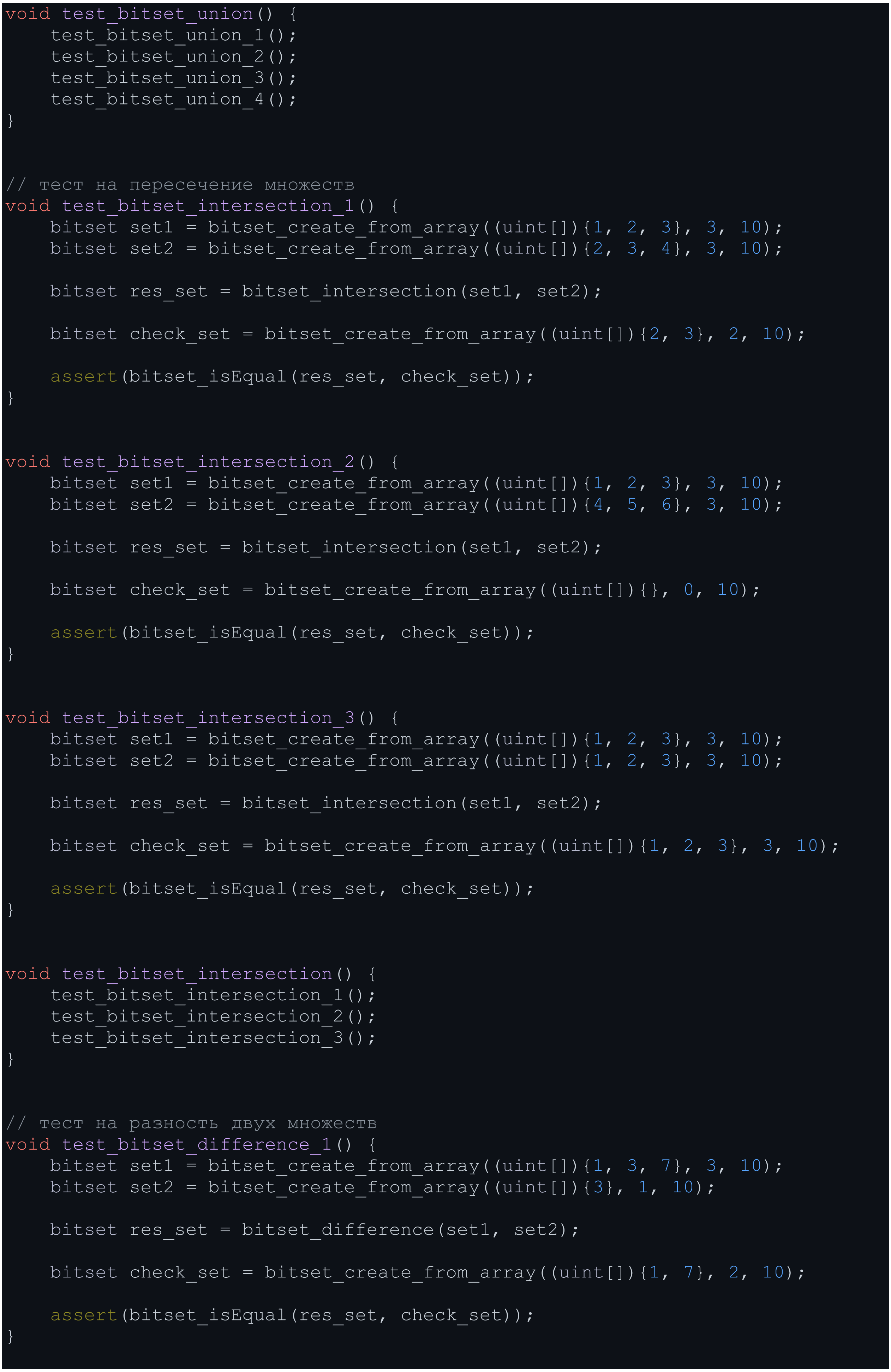
```c
#include <stdio.h>
#include "libs/algorithms/array/array.h"
#include "libs/data_structures/bitset/bitset.h"
#include "libs/data_structures/unordered_array_set/unordered_array_set.h"
#include "libs/data_structures/ordered_array_set/ordered_array_set.h"


// тесты с битовыми множествами
typedef unsigned int uint;

// тест на наличие элемента в множестве
void test_bitset_in_1() {
    bitset set = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);
    uint value = 3;

    bool index = bitset_in(set, value);

    assert(index == 1);
}


void test_bitset_in_2() {
    bitset set = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);
    uint value = 4;

    bool index = bitset_in(set, value);

    assert(index == 0);
}


void test_bitset_in() {
    test_bitset_in_1();
    test_bitset_in_2();
}


// тест на является ли множество подмножеством другого множества
void test_bitset_isSubset_1() {
    bitset subset = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);
    bitset set = bitset_create_from_array((uint[]){1, 2, 3, 4, 5, 6}, 6, 10);

    assert(bitset_isSubset(subset, set));
}


void test_bitset_isSubset_2() {
    bitset subset = bitset_create_from_array((uint[]){1, 2, 10}, 3, 10);
    bitset set = bitset_create_from_array((uint[]){1, 2, 3, 4, 5, 6}, 6, 10);

    assert(!bitset_isSubset(subset, set));
}


void test_bitset_isSubset_3() {
    bitset subset = bitset_create_from_array((uint[]){}, 0, 10);
    bitset set = bitset_create_from_array((uint[]){1, 2, 3, 4, 5, 6}, 6, 10);

    assert(bitset_isSubset(subset, set));
}


void test_bitset_isSubset_4() {
```



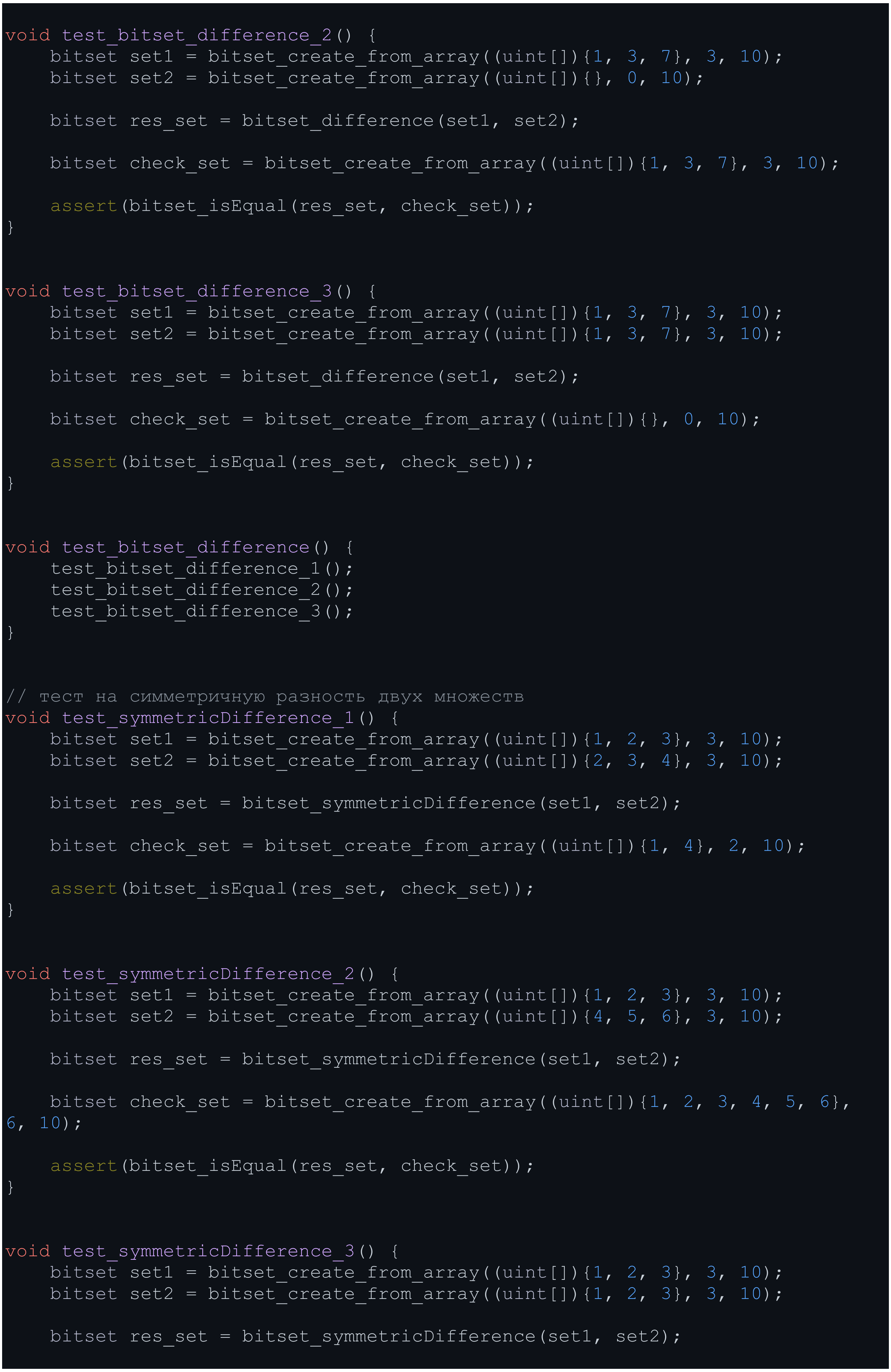
```
    bitset subset = bitset_create_from_array((uint[]){1, 2, 3, 4, 5, 6}, 6,
10);
    bitset set = bitset_create_from_array((uint[]){1, 2, 3, 4, 5, 6}, 6, 10);

    assert(bitset_isSubset(subset, set));
}


void test_bitset_isSubset() {
    test_bitset_isSubset_1();
    test_bitset_isSubset_2();
    test_bitset_isSubset_3();
    test_bitset_isSubset_4();
}


// тест на вставку элемента в множество
void test_bitset_insert_1() {
    bitset set = bitset_create_from_array((uint[]){7, 8}, 2, 10);
    uint value = 4;

    bitset_insert(&set, value);

    bitset check_set = bitset_create_from_array((uint[]){4, 7, 8}, 3, 3);

    assert(bitset_isEqual(set, check_set));
}


void test_bitset_insert_2() {
    bitset set = bitset_create_from_array((uint[]){7, 8}, 2, 10);
    uint value = 7;

    bitset_insert(&set, value);

    bitset check_set = bitset_create_from_array((uint[]){7, 8}, 2, 2);

    assert(bitset_isEqual(set, check_set));
}


void test_bitset_insert() {
    test_bitset_insert_1();
    test_bitset_insert_2();
}


// тест на удаление элемента
void test_bitset_deleteElement_1() {
    bitset set = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);
    uint value = 3;

    bitset_deleteElement(&set, value);

    bitset check_set = bitset_create_from_array((uint[]){1, 2}, 2, 10);

    assert(bitset_isEqual(set, check_set));
}


void test_bitset_deleteElement_2() {
    bitset set = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);
    uint value = 5;
```



```
    bitset_deleteElement(&set, value);

    bitset check_set = bitset_create_from_array((uint[]){1, 2,3}, 3, 2);

    assert(bitset_isEqual(set, check_set));
}

void test_bitset_deleteElement() {
    test_bitset_deleteElement_1();
    test_bitset_deleteElement_2();
}


// тест на объединение множеств
void test_bitset_union_1() {
    bitset set1 = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);
    bitset set2 = bitset_create_from_array((uint[]){3, 4, 5}, 3, 10);

    bitset res_set = bitset_union(set1, set2);

    bitset check_set = bitset_create_from_array((uint[]){1, 2, 3, 4, 5}, 5,
10);

    assert(bitset_isEqual(res_set, check_set));
}


void test_bitset_union_2() {
    bitset set1 = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);
    bitset set2 = bitset_create_from_array((uint[]){4, 5, 6}, 3, 10);

    bitset res_set = bitset_union(set1, set2);

    bitset check_set = bitset_create_from_array((uint[]){1, 2, 3, 4, 5, 6},
6, 10);

    assert(bitset_isEqual(res_set, check_set));
}


void test_bitset_union_3() {
    bitset set1 = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);
    bitset set2 = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);

    bitset res_set = bitset_union(set1, set2);

    bitset check_set = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);

    assert(bitset_isEqual(res_set, check_set));
}


void test_bitset_union_4() {
    bitset set1 = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);
    bitset set2 = bitset_create_from_array((uint[]){}, 0, 10);

    bitset res_set = bitset_union(set1, set2);

    bitset check_set = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);

    assert(bitset_isEqual(res_set, check_set));
}
```

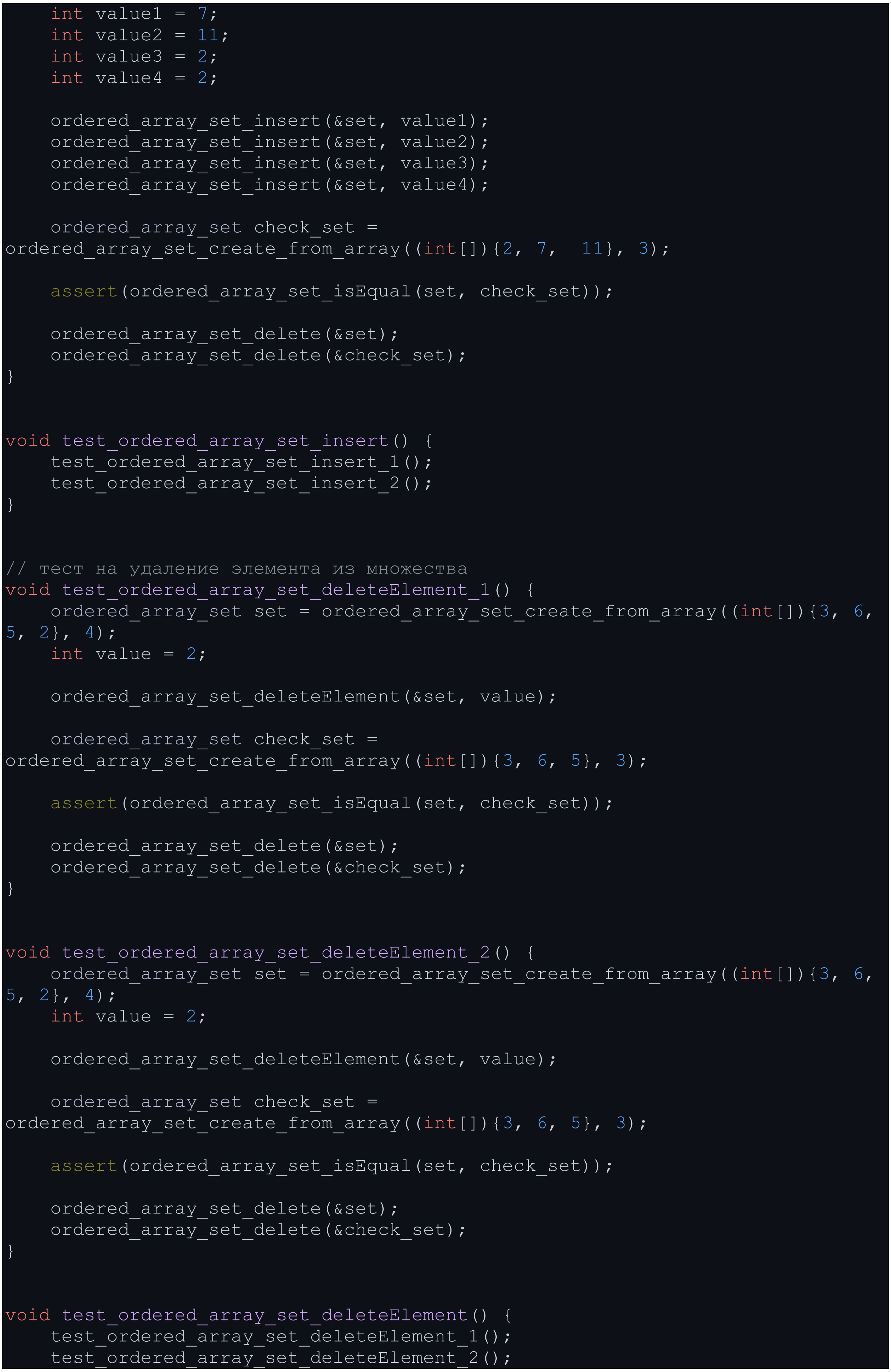


```
void test_bitset_union() {
    test_bitset_union_1();
    test_bitset_union_2();
    test_bitset_union_3();
    test_bitset_union_4();
}


// тест на пересечение множеств
void test_bitset_intersection_1() {
    bitset set1 = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);
    bitset set2 = bitset_create_from_array((uint[]){2, 3, 4}, 3, 10);

    bitset res_set = bitset_intersection(set1, set2);

    bitset check_set = bitset_create_from_array((uint[]){2, 3}, 2, 10);

    assert(bitset_isEqual(res_set, check_set));
}


void test_bitset_intersection_2() {
    bitset set1 = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);
    bitset set2 = bitset_create_from_array((uint[]){4, 5, 6}, 3, 10);

    bitset res_set = bitset_intersection(set1, set2);

    bitset check_set = bitset_create_from_array((uint[]){}, 0, 10);

    assert(bitset_isEqual(res_set, check_set));
}


void test_bitset_intersection_3() {
    bitset set1 = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);
    bitset set2 = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);

    bitset res_set = bitset_intersection(set1, set2);

    bitset check_set = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);

    assert(bitset_isEqual(res_set, check_set));
}


void test_bitset_intersection() {
    test_bitset_intersection_1();
    test_bitset_intersection_2();
    test_bitset_intersection_3();
}


// тест на разность двух множеств
void test_bitset_difference_1() {
    bitset set1 = bitset_create_from_array((uint[]){1, 3, 7}, 3, 10);
    bitset set2 = bitset_create_from_array((uint[]){3}, 1, 10);

    bitset res_set = bitset_difference(set1, set2);

    bitset check_set = bitset_create_from_array((uint[]){1, 7}, 2, 10);

    assert(bitset_isEqual(res_set, check_set));
}
```

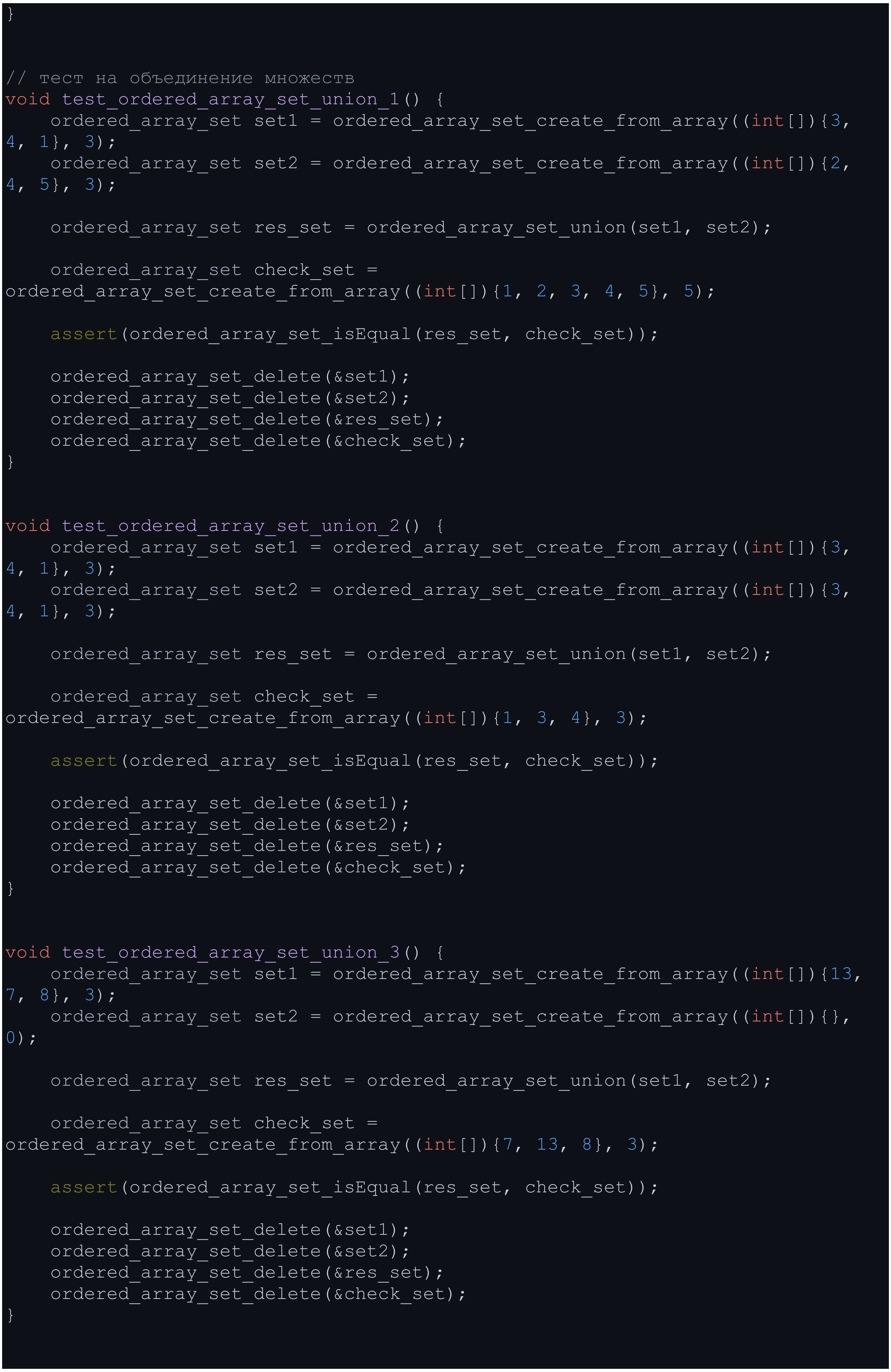


```
void test_bitset_difference_2() {
    bitset set1 = bitset_create_from_array((uint[]){1, 3, 7}, 3, 10);
    bitset set2 = bitset_create_from_array((uint[]){}, 0, 10);

    bitset res_set = bitset_difference(set1, set2);

    bitset check_set = bitset_create_from_array((uint[]){1, 3, 7}, 3, 10);

    assert(bitset_isEqual(res_set, check_set));
}


void test_bitset_difference_3() {
    bitset set1 = bitset_create_from_array((uint[]){1, 3, 7}, 3, 10);
    bitset set2 = bitset_create_from_array((uint[]){1, 3, 7}, 3, 10);

    bitset res_set = bitset_difference(set1, set2);

    bitset check_set = bitset_create_from_array((uint[]){}, 0, 10);

    assert(bitset_isEqual(res_set, check_set));
}


void test_bitset_difference() {
    test_bitset_difference_1();
    test_bitset_difference_2();
    test_bitset_difference_3();
}


// тест на симметричную разность двух множеств
void test_symmetricDifference_1() {
    bitset set1 = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);
    bitset set2 = bitset_create_from_array((uint[]){2, 3, 4}, 3, 10);

    bitset res_set = bitset_symmetricDifference(set1, set2);

    bitset check_set = bitset_create_from_array((uint[]){1, 4}, 2, 10);

    assert(bitset_isEqual(res_set, check_set));
}


void test_symmetricDifference_2() {
    bitset set1 = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);
    bitset set2 = bitset_create_from_array((uint[]){4, 5, 6}, 3, 10);

    bitset res_set = bitset_symmetricDifference(set1, set2);

    bitset check_set = bitset_create_from_array((uint[]){1, 2, 3, 4, 5, 6},
6, 10);

    assert(bitset_isEqual(res_set, check_set));
}


void test_symmetricDifference_3() {
    bitset set1 = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);
    bitset set2 = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);

    bitset res_set = bitset_symmetricDifference(set1, set2);
    bitset check_set = bitset_create_from_array((uint[]){}, 0, 10);

    assert(bitset_isEqual(res_set, check_set));
}


void test_symmetricDifference() {
    test_symmetricDifference_1();
    test_symmetricDifference_2();
    test_symmetricDifference_3();
}


// тест на дополнение множества
void test_bitset_complement_1() {
    bitset set = bitset_create_from_array((uint[]){1, 2, 3}, 3, 10);

    bitset res_set = bitset_complement(set);

    bitset check_set = bitset_create_from_array((uint[]){0, 4, 5, 6, 7, 8, 9,
10}, 8, 10);

    assert(bitset_isEqual(res_set, check_set));
}


void test_bitset_complement_2() {
    bitset set = bitset_create_from_array((uint[]){}, 0, 10);

    bitset res_set = bitset_complement(set);

    bitset check_set = bitset_create_from_array((uint[]){0, 1, 2, 3,4, 5, 6,
7, 8, 9, 10}, 11, 12);

    assert(bitset_isEqual(res_set, check_set));
}


void test_bitset_complement_3() {
    bitset set = bitset_create_from_array((uint[]){0, 1, 2, 3,4, 5, 6, 7, 8,
9, 10}, 11, 10);

    bitset res_set = bitset_complement(set);

    bitset check_set = bitset_create_from_array((uint[]){}, 0, 10);

    assert(bitset_isEqual(res_set, check_set));
}



void test_bitset_complement() {
    test_bitset_complement_1();
    test_bitset_complement_2();
    test_bitset_complement_3();
}


void test_bitset() {
    test_bitset_in();
    test_bitset_isSubset();
    test_bitset_insert();
    test_bitset_deleteElement();
    test_bitset_union();
    test_bitset_intersection();
    test_bitset_difference();
    test_symmetricDifference();
    test_bitset_complement();
}


// тест для упорядоченных множеств



// тест на наличие элемента в множестве
void test_ordered_array_set_in_1() {
    ordered_array_set set = ordered_array_set_create_from_array((int[]){1, 2,
3}, 3);
    int value = 3;

    size_t index = ordered_array_set_in(&set, value);

    assert(index == 2);

    ordered_array_set_delete(&set);
}


void test_ordered_array_set_in_2() {
    ordered_array_set set = ordered_array_set_create_from_array((int[]){1, 2,
3}, 3);
    int value = 5;

    size_t index = ordered_array_set_in(&set, value);

    assert(index == 3);

    ordered_array_set_delete(&set);
}


void test_ordered_array_set_in() {
    test_ordered_array_set_in_1();
    test_ordered_array_set_in_2();
}


// тест является ли множество подномножеством другого множества
void test_ordered_array_set_isSubset_1() {
    ordered_array_set subset = ordered_array_set_create_from_array((int[]){1,
2, 3}, 3);
    ordered_array_set set = ordered_array_set_create_from_array((int[]){1, 2,
3, 4, 5, 6}, 6);

    assert(ordered_array_set_isSubset(subset, set));

    ordered_array_set_delete(&subset);
    ordered_array_set_delete(&set);
}


void test_ordered_array_set_isSubset_2() {
    ordered_array_set subset = ordered_array_set_create_from_array((int[]){4,
5, 3}, 3);
    ordered_array_set set = ordered_array_set_create_from_array((int[]){1, 2,
3, 4, 5, 6}, 6);
```

```c
    assert(ordered_array_set_isSubset(subset, set));

    ordered_array_set_delete(&subset);
    ordered_array_set_delete(&set);
}


void test_ordered_array_set_isSubset_3() {
    ordered_array_set subset = ordered_array_set_create_from_array((int[]){5,
3, 2, 4, 1, 6}, 6);
    ordered_array_set set = ordered_array_set_create_from_array((int[]){1, 2,
3, 4, 5, 6}, 6);

    assert(ordered_array_set_isSubset(subset, set));

    ordered_array_set_delete(&subset);
    ordered_array_set_delete(&set);
}


void test_ordered_array_set_isSubset_4() {
    ordered_array_set subset =
ordered_array_set_create_from_array((int[]){10}, 1);
    ordered_array_set set = ordered_array_set_create_from_array((int[]){1, 2,
3, 4, 5, 6}, 6);

    assert(!ordered_array_set_isSubset(subset, set));

    ordered_array_set_delete(&subset);
    ordered_array_set_delete(&set);
}


void test_ordered_array_set_isSubset() {
    test_ordered_array_set_isSubset_1();
    test_ordered_array_set_isSubset_2();
    test_ordered_array_set_isSubset_3();
    test_ordered_array_set_isSubset_4();
}


// тест на вставку элемента в множество
void test_ordered_array_set_insert_1() {
    ordered_array_set set = ordered_array_set_create(10);
    int value1 = 2;
    int value2 = 1;
    int value3 = 3;

    ordered_array_set_insert(&set, value1);
    ordered_array_set_insert(&set, value2);
    ordered_array_set_insert(&set, value3);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){1, 2, 3}, 3);

    assert(ordered_array_set_isEqual(set, check_set));

    ordered_array_set_delete(&set);
    ordered_array_set_delete(&check_set);
}


void test_ordered_array_set_insert_2() {
    ordered_array_set set = ordered_array_set_create(10);
```

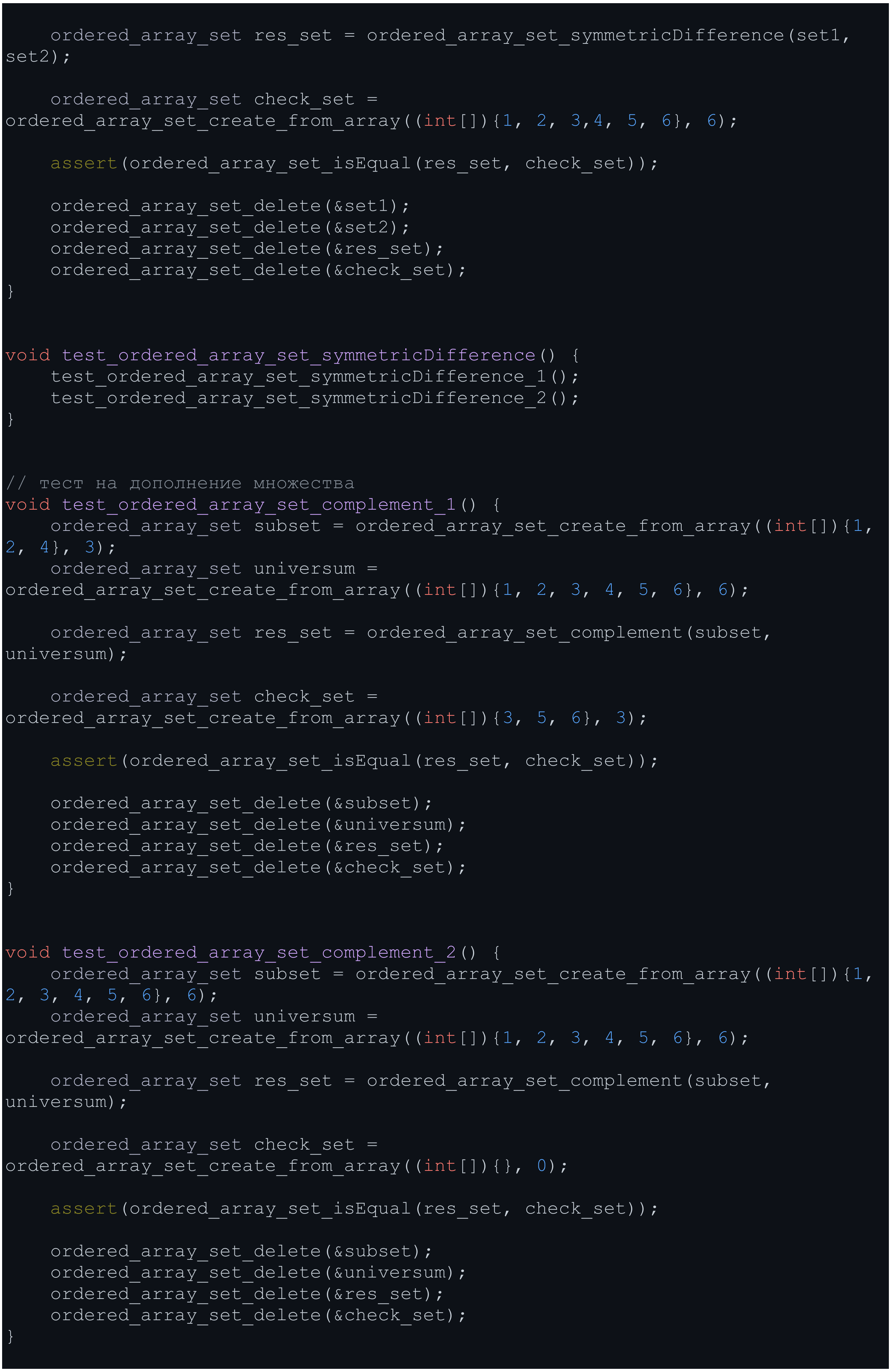


```
    int value1 = 7;
    int value2 = 11;
    int value3 = 2;
    int value4 = 2;

    ordered_array_set_insert(&set, value1);
    ordered_array_set_insert(&set, value2);
    ordered_array_set_insert(&set, value3);
    ordered_array_set_insert(&set, value4);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){2, 7,  11}, 3);

    assert(ordered_array_set_isEqual(set, check_set));

    ordered_array_set_delete(&set);
    ordered_array_set_delete(&check_set);
}


void test_ordered_array_set_insert() {
    test_ordered_array_set_insert_1();
    test_ordered_array_set_insert_2();
}


// тест на удаление элемента из множества
void test_ordered_array_set_deleteElement_1() {
    ordered_array_set set = ordered_array_set_create_from_array((int[]){3, 6,
5, 2}, 4);
    int value = 2;

    ordered_array_set_deleteElement(&set, value);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){3, 6, 5}, 3);

    assert(ordered_array_set_isEqual(set, check_set));

    ordered_array_set_delete(&set);
    ordered_array_set_delete(&check_set);
}


void test_ordered_array_set_deleteElement_2() {
    ordered_array_set set = ordered_array_set_create_from_array((int[]){3, 6,
5, 2}, 4);
    int value = 2;

    ordered_array_set_deleteElement(&set, value);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){3, 6, 5}, 3);

    assert(ordered_array_set_isEqual(set, check_set));

    ordered_array_set_delete(&set);
    ordered_array_set_delete(&check_set);
}


void test_ordered_array_set_deleteElement() {
    test_ordered_array_set_deleteElement_1();
    test_ordered_array_set_deleteElement_2();
```



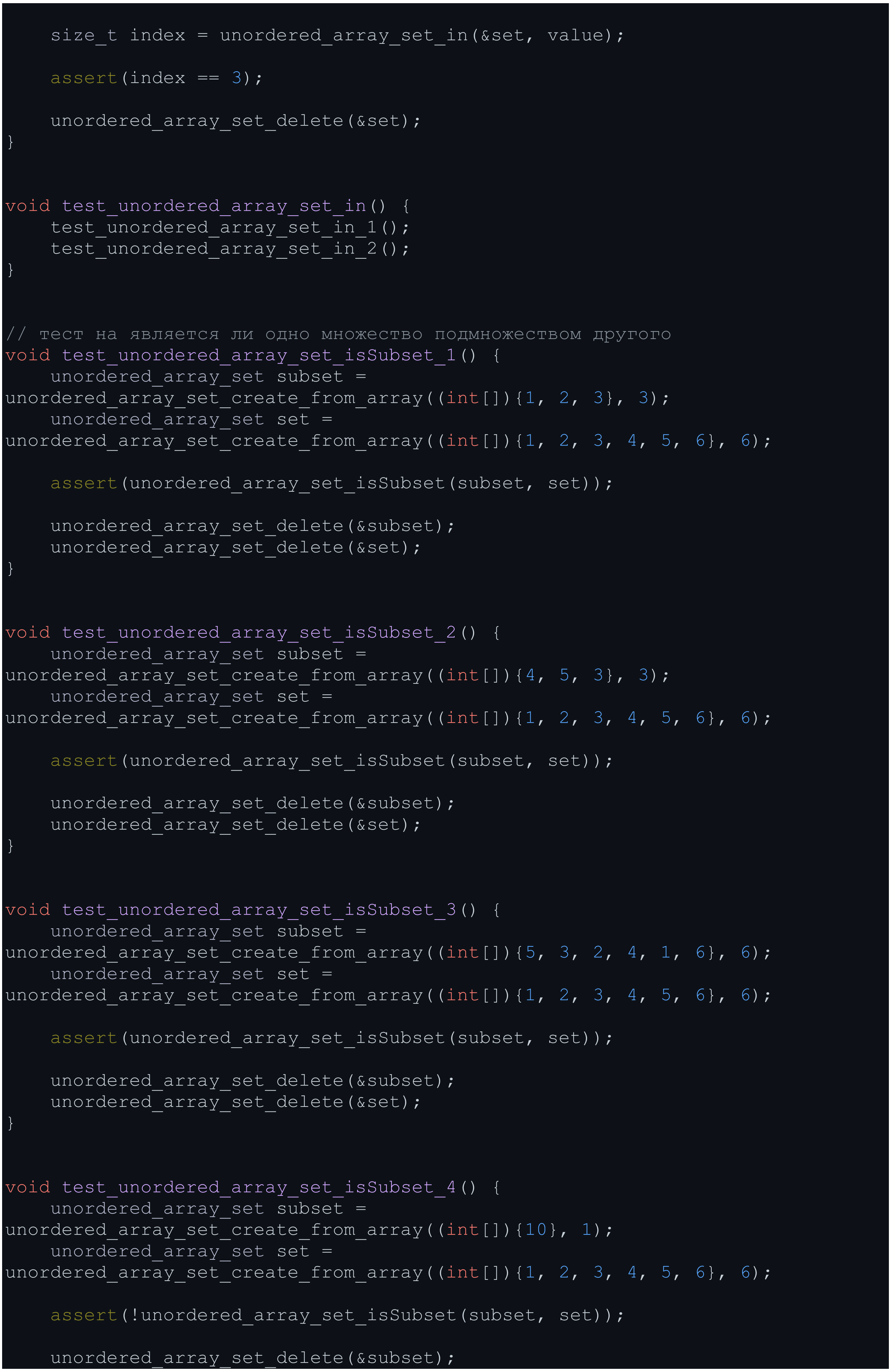
```
}


// тест на объединение множеств
void test_ordered_array_set_union_1() {
    ordered_array_set set1 = ordered_array_set_create_from_array((int[]){3,
4, 1}, 3);
    ordered_array_set set2 = ordered_array_set_create_from_array((int[]){2,
4, 5}, 3);

    ordered_array_set res_set = ordered_array_set_union(set1, set2);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){1, 2, 3, 4, 5}, 5);

    assert(ordered_array_set_isEqual(res_set, check_set));

    ordered_array_set_delete(&set1);
    ordered_array_set_delete(&set2);
    ordered_array_set_delete(&res_set);
    ordered_array_set_delete(&check_set);
}


void test_ordered_array_set_union_2() {
    ordered_array_set set1 = ordered_array_set_create_from_array((int[]){3,
4, 1}, 3);
    ordered_array_set set2 = ordered_array_set_create_from_array((int[]){3,
4, 1}, 3);

    ordered_array_set res_set = ordered_array_set_union(set1, set2);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){1, 3, 4}, 3);

    assert(ordered_array_set_isEqual(res_set, check_set));

    ordered_array_set_delete(&set1);
    ordered_array_set_delete(&set2);
    ordered_array_set_delete(&res_set);
    ordered_array_set_delete(&check_set);
}


void test_ordered_array_set_union_3() {
    ordered_array_set set1 = ordered_array_set_create_from_array((int[]){13,
7, 8}, 3);
    ordered_array_set set2 = ordered_array_set_create_from_array((int[]){},
0);

    ordered_array_set res_set = ordered_array_set_union(set1, set2);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){7, 13, 8}, 3);

    assert(ordered_array_set_isEqual(res_set, check_set));

    ordered_array_set_delete(&set1);
    ordered_array_set_delete(&set2);
    ordered_array_set_delete(&res_set);
    ordered_array_set_delete(&check_set);
}
```

```
void test_ordered_array_set_union() {
    test_ordered_array_set_union_1();
    test_ordered_array_set_union_2();
    test_ordered_array_set_union_3();
}


// тест на пересечение двух множеств
void test_ordered_array_set_intersection_1() {
    ordered_array_set set1 = ordered_array_set_create_from_array((int[]){1,
2, 3}, 3);
    ordered_array_set set2 = ordered_array_set_create_from_array((int[]){2,
3, 4}, 3);

    ordered_array_set res_set = ordered_array_set_intersection(set1, set2);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){2, 3}, 2);

    assert(ordered_array_set_isEqual(res_set, check_set));

    ordered_array_set_delete(&set1);
    ordered_array_set_delete(&set2);
    ordered_array_set_delete(&res_set);
    ordered_array_set_delete(&check_set);
}


void test_ordered_array_set_intersection_2() {
    ordered_array_set set1 = ordered_array_set_create_from_array((int[]){1,
2, 3}, 3);
    ordered_array_set set2 = ordered_array_set_create_from_array((int[]){4,
5, 6}, 3);

    ordered_array_set res_set = ordered_array_set_intersection(set1, set2);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){}, 0);

    assert(ordered_array_set_isEqual(res_set, check_set));

    ordered_array_set_delete(&set1);
    ordered_array_set_delete(&set2);
    ordered_array_set_delete(&res_set);
    ordered_array_set_delete(&check_set);
}


void test_ordered_array_set_intersection() {
    test_ordered_array_set_intersection_1();
    test_ordered_array_set_intersection_2();
}


// тест на разность множеств
void test_ordered_array_set_difference_1() {
    ordered_array_set set1 = ordered_array_set_create_from_array((int[]){1,
2, 3, 4, 5, 6}, 6);
    ordered_array_set set2 = ordered_array_set_create_from_array((int[]){2,
3, 6}, 3);

    ordered_array_set res_set = ordered_array_set_difference(set1, set2);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){1, 4, 5}, 3);

    assert(ordered_array_set_isEqual(res_set, check_set));

    ordered_array_set_delete(&set1);
    ordered_array_set_delete(&set2);
    ordered_array_set_delete(&res_set);
    ordered_array_set_delete(&check_set);
}


void test_ordered_array_set_difference_2() {
    ordered_array_set set1 = ordered_array_set_create_from_array((int[]){1,
2, 3, 4, 5, 6}, 6);
    ordered_array_set set2 = ordered_array_set_create_from_array((int[]){7,
8, 9}, 3);

    ordered_array_set res_set = ordered_array_set_difference(set1, set2);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){1, 2, 3, 4, 5, 6}, 6);

    assert(ordered_array_set_isEqual(res_set, check_set));

    ordered_array_set_delete(&set1);
    ordered_array_set_delete(&set2);
    ordered_array_set_delete(&res_set);
    ordered_array_set_delete(&check_set);
}


void test_ordered_array_set_difference() {
    test_ordered_array_set_difference_1();
    test_ordered_array_set_difference_2();
}


// тест на симметричную разность
void test_ordered_array_set_symmetricDifference_1() {
    ordered_array_set set1 = ordered_array_set_create_from_array((int[]){1,
2, 3, 4, 5, 6}, 6);
    ordered_array_set set2 = ordered_array_set_create_from_array((int[]){2,
3, 6, 7, 10, 12}, 6);

    ordered_array_set res_set = ordered_array_set_symmetricDifference(set1,
set2);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){1, 4, 5,7, 10, 12}, 6);

    assert(ordered_array_set_isEqual(res_set, check_set));

    ordered_array_set_delete(&set1);
    ordered_array_set_delete(&set2);
    ordered_array_set_delete(&res_set);
    ordered_array_set_delete(&check_set);
}


void test_ordered_array_set_symmetricDifference_2() {
    ordered_array_set set1 = ordered_array_set_create_from_array((int[]){1,
2, 3}, 3);
    ordered_array_set set2 = ordered_array_set_create_from_array((int[]){4,
5, 6}, 3);
```

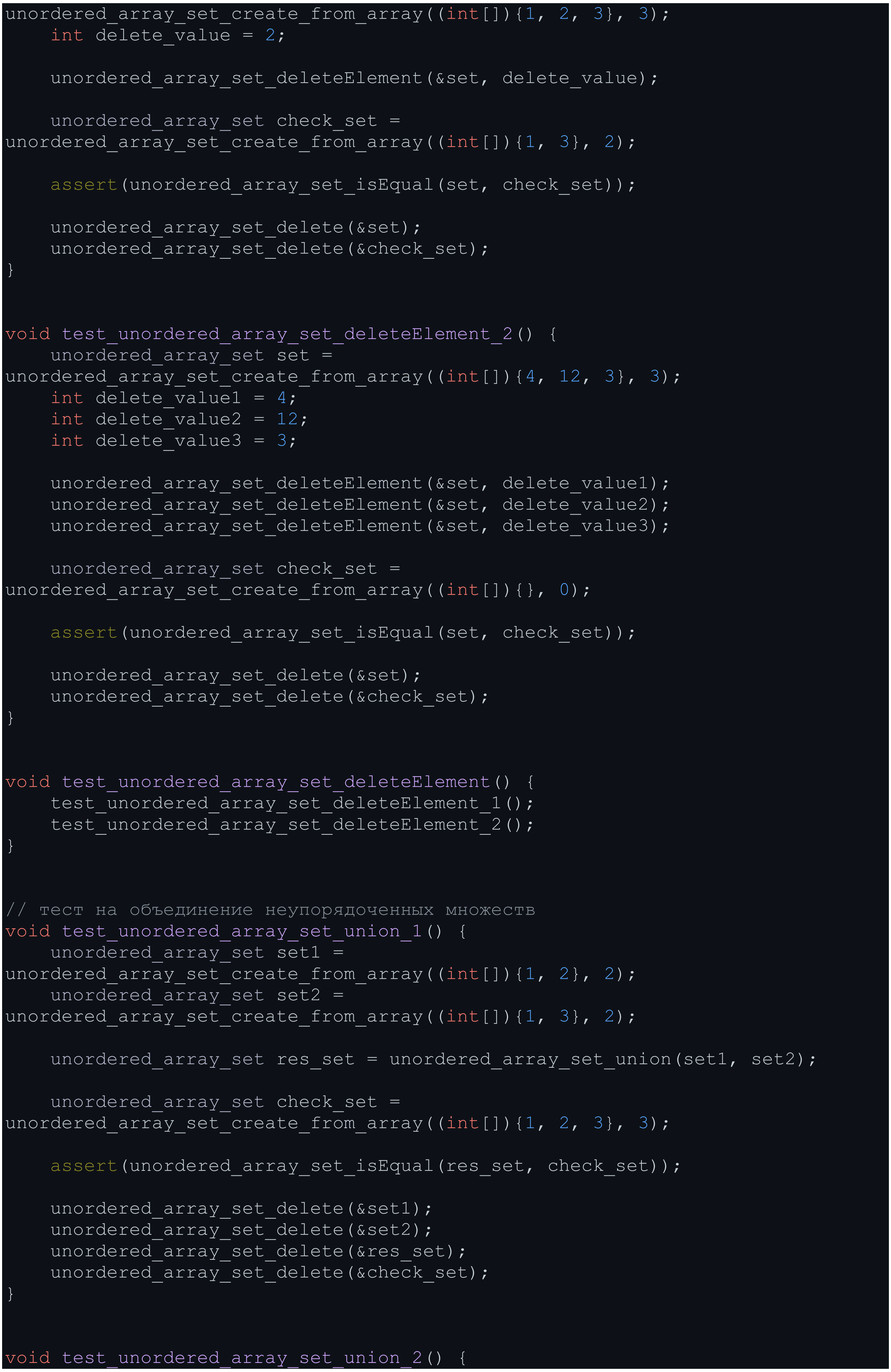


```
    ordered_array_set res_set = ordered_array_set_symmetricDifference(set1,
set2);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){1, 2, 3,4, 5, 6}, 6);

    assert(ordered_array_set_isEqual(res_set, check_set));

    ordered_array_set_delete(&set1);
    ordered_array_set_delete(&set2);
    ordered_array_set_delete(&res_set);
    ordered_array_set_delete(&check_set);
}


void test_ordered_array_set_symmetricDifference() {
    test_ordered_array_set_symmetricDifference_1();
    test_ordered_array_set_symmetricDifference_2();
}


// тест на дополнение множества
void test_ordered_array_set_complement_1() {
    ordered_array_set subset = ordered_array_set_create_from_array((int[]){1,
2, 4}, 3);
    ordered_array_set universum =
ordered_array_set_create_from_array((int[]){1, 2, 3, 4, 5, 6}, 6);

    ordered_array_set res_set = ordered_array_set_complement(subset,
universum);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){3, 5, 6}, 3);

    assert(ordered_array_set_isEqual(res_set, check_set));

    ordered_array_set_delete(&subset);
    ordered_array_set_delete(&universum);
    ordered_array_set_delete(&res_set);
    ordered_array_set_delete(&check_set);
}


void test_ordered_array_set_complement_2() {
    ordered_array_set subset = ordered_array_set_create_from_array((int[]){1,
2, 3, 4, 5, 6}, 6);
    ordered_array_set universum =
ordered_array_set_create_from_array((int[]){1, 2, 3, 4, 5, 6}, 6);

    ordered_array_set res_set = ordered_array_set_complement(subset,
universum);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){}, 0);

    assert(ordered_array_set_isEqual(res_set, check_set));

    ordered_array_set_delete(&subset);
    ordered_array_set_delete(&universum);
    ordered_array_set_delete(&res_set);
    ordered_array_set_delete(&check_set);
}
```

```
void test_ordered_array_set_complement_3() {
    ordered_array_set subset = ordered_array_set_create_from_array((int[]){},
0);
    ordered_array_set universum =
ordered_array_set_create_from_array((int[]){1, 2, 3, 4, 5, 6}, 6);

    ordered_array_set res_set = ordered_array_set_complement(subset,
universum);

    ordered_array_set check_set =
ordered_array_set_create_from_array((int[]){1, 2, 3, 4, 5, 6}, 6);

    assert(ordered_array_set_isEqual(res_set, check_set));

    ordered_array_set_delete(&subset);
    ordered_array_set_delete(&universum);
    ordered_array_set_delete(&res_set);
    ordered_array_set_delete(&check_set);
}


void test_ordered_array_set_complement() {
    test_ordered_array_set_complement_1();
    test_ordered_array_set_complement_2();
    test_ordered_array_set_complement_3();
}


void test_ordered_array_set() {
    test_ordered_array_set_in();
    test_ordered_array_set_isSubset();
    test_ordered_array_set_insert();
    test_ordered_array_set_deleteElement();
    test_ordered_array_set_union();
    test_ordered_array_set_intersection();
    test_ordered_array_set_difference();
    test_ordered_array_set_symmetricDifference();
    test_ordered_array_set_complement();
}


// тесты для неупорядоченного множества



// тест на наличие элемента в множестве
void test_unordered_array_set_in_1() {
    unordered_array_set set =
unordered_array_set_create_from_array((int[]){1, 2, 3}, 3);
    int value = 2;

    size_t index = unordered_array_set_in(&set, value);

    assert(index == 1);

    unordered_array_set_delete(&set);
}


void test_unordered_array_set_in_2() {
    unordered_array_set set =
unordered_array_set_create_from_array((int[]){10, 1, 4}, 3);
    int value = 5;
```



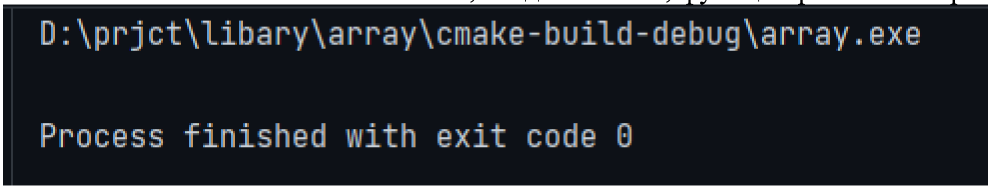
```c
    size_t index = unordered_array_set_in(&set, value);

    assert(index == 3);

    unordered_array_set_delete(&set);
}


void test_unordered_array_set_in() {
    test_unordered_array_set_in_1();
    test_unordered_array_set_in_2();
}


// тест на является ли одно множество подмножеством другого
void test_unordered_array_set_isSubset_1() {
    unordered_array_set subset =
unordered_array_set_create_from_array((int[]){1, 2, 3}, 3);
    unordered_array_set set =
unordered_array_set_create_from_array((int[]){1, 2, 3, 4, 5, 6}, 6);

    assert(unordered_array_set_isSubset(subset, set));

    unordered_array_set_delete(&subset);
    unordered_array_set_delete(&set);
}


void test_unordered_array_set_isSubset_2() {
    unordered_array_set subset =
unordered_array_set_create_from_array((int[]){4, 5, 3}, 3);
    unordered_array_set set =
unordered_array_set_create_from_array((int[]){1, 2, 3, 4, 5, 6}, 6);

    assert(unordered_array_set_isSubset(subset, set));

    unordered_array_set_delete(&subset);
    unordered_array_set_delete(&set);
}


void test_unordered_array_set_isSubset_3() {
    unordered_array_set subset =
unordered_array_set_create_from_array((int[]){5, 3, 2, 4, 1, 6}, 6);
    unordered_array_set set =
unordered_array_set_create_from_array((int[]){1, 2, 3, 4, 5, 6}, 6);

    assert(unordered_array_set_isSubset(subset, set));

    unordered_array_set_delete(&subset);
    unordered_array_set_delete(&set);
}


void test_unordered_array_set_isSubset_4() {
    unordered_array_set subset =
unordered_array_set_create_from_array((int[]){10}, 1);
    unordered_array_set set =
unordered_array_set_create_from_array((int[]){1, 2, 3, 4, 5, 6}, 6);

    assert(!unordered_array_set_isSubset(subset, set));

    unordered_array_set_delete(&subset);
    unordered_array_set_delete(&set);
}


void test_unordered_array_set_isSubset() {
    test_unordered_array_set_isSubset_1();
    test_unordered_array_set_isSubset_2();
    test_unordered_array_set_isSubset_3();
    test_unordered_array_set_isSubset_4();
}


// тест на вставку элемента
void test_unordered_array_set_insert_1() {
    unordered_array_set set = unordered_array_set_create(10);
    int value1 = 1;
    int value2 = 7;
    int value3 = 5;

    unordered_array_set_insert(&set, value1);
    unordered_array_set_insert(&set, value2);
    unordered_array_set_insert(&set, value3);

    unordered_array_set check_set =
unordered_array_set_create_from_array((int[]){1, 7, 5}, 3);

    assert(unordered_array_set_isEqual(set, check_set));

    unordered_array_set_delete(&set);
    unordered_array_set_delete(&check_set);
}


void test_unordered_array_set_insert_2() {
    unordered_array_set set = unordered_array_set_create(10);
    int value1 = 3;
    int value2 = 7;
    int value3 = 7;
    int value4 = 8;

    unordered_array_set_insert(&set, value1);
    unordered_array_set_insert(&set, value2);
    unordered_array_set_insert(&set, value3);
    unordered_array_set_insert(&set, value4);

    unordered_array_set check_set =
unordered_array_set_create_from_array((int[]){3, 7, 8}, 3);

    assert(unordered_array_set_isEqual(set, check_set));

    unordered_array_set_delete(&set);
    unordered_array_set_delete(&check_set);
}


void test_unordered_array_set_insert() {
    test_unordered_array_set_insert_1();
    test_unordered_array_set_insert_2();
}


// тест на удаление элемента из множества
void test_unordered_array_set_deleteElement_1() {
    unordered_array_set set =
```



```
unordered_array_set_create_from_array((int[]){1, 2, 3}, 3);
    int delete_value = 2;

    unordered_array_set_deleteElement(&set, delete_value);

    unordered_array_set check_set =
unordered_array_set_create_from_array((int[]){1, 3}, 2);

    assert(unordered_array_set_isEqual(set, check_set));

    unordered_array_set_delete(&set);
    unordered_array_set_delete(&check_set);
}


void test_unordered_array_set_deleteElement_2() {
    unordered_array_set set =
unordered_array_set_create_from_array((int[]){4, 12, 3}, 3);
    int delete_value1 = 4;
    int delete_value2 = 12;
    int delete_value3 = 3;

    unordered_array_set_deleteElement(&set, delete_value1);
    unordered_array_set_deleteElement(&set, delete_value2);
    unordered_array_set_deleteElement(&set, delete_value3);

    unordered_array_set check_set =
unordered_array_set_create_from_array((int[]){}, 0);

    assert(unordered_array_set_isEqual(set, check_set));

    unordered_array_set_delete(&set);
    unordered_array_set_delete(&check_set);
}


void test_unordered_array_set_deleteElement() {
    test_unordered_array_set_deleteElement_1();
    test_unordered_array_set_deleteElement_2();
}


// тест на объединение неупорядоченных множеств
void test_unordered_array_set_union_1() {
    unordered_array_set set1 =
unordered_array_set_create_from_array((int[]){1, 2}, 2);
    unordered_array_set set2 =
unordered_array_set_create_from_array((int[]){1, 3}, 2);

    unordered_array_set res_set = unordered_array_set_union(set1, set2);

    unordered_array_set check_set =
unordered_array_set_create_from_array((int[]){1, 2, 3}, 3);

    assert(unordered_array_set_isEqual(res_set, check_set));

    unordered_array_set_delete(&set1);
    unordered_array_set_delete(&set2);
    unordered_array_set_delete(&res_set);
    unordered_array_set_delete(&check_set);
}


void test_unordered_array_set_union_2() {
    unordered_array_set set1 =
unordered_array_set_create_from_array((int[]){5, 7, 8}, 3);
    unordered_array_set set2 =
unordered_array_set_create_from_array((int[]){}, 0);

    unordered_array_set res_set = unordered_array_set_union(set1, set2);

    unordered_array_set check_set =
unordered_array_set_create_from_array((int[]){5, 7, 8}, 3);

    assert(unordered_array_set_isEqual(res_set, check_set));

    unordered_array_set_delete(&set1);
    unordered_array_set_delete(&set2);
    unordered_array_set_delete(&res_set);
    unordered_array_set_delete(&check_set);
}


void test_unordered_array_set_union() {
    test_unordered_array_set_union_1();
    test_unordered_array_set_union_2();
}


// тест на пересечение множеств
void test_unordered_array_set_intersection_1() {
    unordered_array_set set1 =
unordered_array_set_create_from_array((int[]){1 , 3, 4}, 3);
    unordered_array_set set2 = unordered_array_set_create_from_array((int[]){
3, 4, 5}, 3);

    unordered_array_set res_set = unordered_array_set_intersection(set1,
set2);

    unordered_array_set check_set =
unordered_array_set_create_from_array((int[]){3, 4}, 2);

    assert(unordered_array_set_isEqual(res_set, check_set));

    unordered_array_set_delete(&set1);
    unordered_array_set_delete(&set2);
    unordered_array_set_delete(&res_set);
    unordered_array_set_delete(&check_set);
}


void test_unordered_array_set_intersection_2() {
    unordered_array_set set1 =
unordered_array_set_create_from_array((int[]){1 , 2, 3}, 3);
    unordered_array_set set2 = unordered_array_set_create_from_array((int[]){
4, 5, 6}, 3);

    unordered_array_set res_set = unordered_array_set_intersection(set1,
set2);

    unordered_array_set check_set =
unordered_array_set_create_from_array((int[]){}, 0);

    assert(unordered_array_set_isEqual(res_set, check_set));

    unordered_array_set_delete(&set1);
    unordered_array_set_delete(&set2);
    unordered_array_set_delete(&res_set);
    unordered_array_set_delete(&check_set);
}


void test_unordered_array_set_intersection() {
    test_unordered_array_set_intersection_1();
    test_unordered_array_set_intersection_2();
}


// тест на вычитание одного множества из другого
void test_unordered_array_set_difference_1() {
    unordered_array_set set1 =
unordered_array_set_create_from_array((int[]){1 , 2, 3, 4, 5, 6}, 6);
    unordered_array_set set2 = unordered_array_set_create_from_array((int[]){
4, 5, 6}, 3);

    unordered_array_set res_set = unordered_array_set_difference(set1, set2);

    unordered_array_set check_set =
unordered_array_set_create_from_array((int[]){1, 2, 3}, 3);

    assert(unordered_array_set_isEqual(res_set, check_set));

    unordered_array_set_delete(&set1);
    unordered_array_set_delete(&set2);
    unordered_array_set_delete(&res_set);
    unordered_array_set_delete(&check_set);
}


void test_unordered_array_set_difference_2() {
    unordered_array_set set1 =
unordered_array_set_create_from_array((int[]){1 , 2, 3, 4, 5, 6}, 6);
    unordered_array_set set2 = unordered_array_set_create_from_array((int[]){
7}, 1);

    unordered_array_set res_set = unordered_array_set_difference(set1, set2);

    unordered_array_set check_set =
unordered_array_set_create_from_array((int[]){1, 2, 3, 4, 5, 6}, 6);

    assert(unordered_array_set_isEqual(res_set, check_set));

    unordered_array_set_delete(&set1);
    unordered_array_set_delete(&set2);
    unordered_array_set_delete(&res_set);
    unordered_array_set_delete(&check_set);
}


void test_unordered_array_set_difference_3() {
    unordered_array_set set1 =
unordered_array_set_create_from_array((int[]){1 , 2, 3}, 3);
    unordered_array_set set2 = unordered_array_set_create_from_array((int[]){
3, 2, 1}, 3);

    unordered_array_set res_set = unordered_array_set_difference(set1, set2);

    unordered_array_set check_set =
unordered_array_set_create_from_array((int[]){}, 0);

    assert(unordered_array_set_isEqual(res_set, check_set));
    unordered_array_set_delete(&set1);
    unordered_array_set_delete(&set2);
    unordered_array_set_delete(&res_set);
    unordered_array_set_delete(&check_set);
}


void test_unordered_array_set_difference() {
    test_unordered_array_set_difference_1();
    test_unordered_array_set_difference_2();
    test_unordered_array_set_difference_3();
}


// тест на симметричную разность двух множеств
void test_unordered_array_set_symmetricDifference_1() {
    unordered_array_set set1 =
unordered_array_set_create_from_array((int[]){1 , 3, 4}, 3);
    unordered_array_set set2 = unordered_array_set_create_from_array((int[]){
3, 4, 5}, 3);

    unordered_array_set res_set =
unordered_array_set_symmetricDifference(set1, set2);

    unordered_array_set check_set =
unordered_array_set_create_from_array((int[]){1, 5}, 2);

    assert(unordered_array_set_isEqual(res_set, check_set));

    unordered_array_set_delete(&set1);
    unordered_array_set_delete(&set2);
    unordered_array_set_delete(&res_set);
    unordered_array_set_delete(&check_set);
}


void test_unordered_array_set_symmetricDifference_2() {
    unordered_array_set set1 =
unordered_array_set_create_from_array((int[]){1 , 2, 3}, 3);
    unordered_array_set set2 = unordered_array_set_create_from_array((int[]){
4, 5, 6}, 3);

    unordered_array_set res_set =
unordered_array_set_symmetricDifference(set1, set2);

    unordered_array_set check_set =
unordered_array_set_create_from_array((int[]){1, 2, 3, 4, 6, 5}, 6);

    assert(unordered_array_set_isEqual(res_set, check_set));

    unordered_array_set_delete(&set1);
    unordered_array_set_delete(&set2);
    unordered_array_set_delete(&res_set);
    unordered_array_set_delete(&check_set);
}


void test_unordered_array_set_symmetricDifference() {
    test_unordered_array_set_symmetricDifference_1();
    test_unordered_array_set_symmetricDifference_2();
}


// тест на дополнение множества до универсума
```

```c
void test_unordered_array_set_complement_1() {
    unordered_array_set set = unordered_array_set_create_from_array((int[]){1
,2, 3}, 3);
    unordered_array_set universum =
unordered_array_set_create_from_array((int[]){1, 2, 3, 4, 5, 6}, 6);

    unordered_array_set res_set = unordered_array_set_complement(set,
universum);

    assert(unordered_array_set_isSubset(res_set, universum));

    unordered_array_set_delete(&set);
    unordered_array_set_delete(&universum);
    unordered_array_set_delete(&res_set);
}


void test_unordered_array_set_complement_2() {
    unordered_array_set set = unordered_array_set_create_from_array((int[]){1
,2, 3, 4, 5, 6}, 6);
    unordered_array_set universum =
unordered_array_set_create_from_array((int[]){1, 2, 3, 4, 5, 6}, 6);

    unordered_array_set res_set = unordered_array_set_complement(set,
universum);

    assert(unordered_array_set_isSubset(res_set, universum));

    unordered_array_set_delete(&set);
    unordered_array_set_delete(&universum);
    unordered_array_set_delete(&res_set);
}


void test_unordered_array_set_complement_3() {
    unordered_array_set set =
unordered_array_set_create_from_array((int[]){}, 0);
    unordered_array_set universum =
unordered_array_set_create_from_array((int[]){1, 2, 3, 4, 5, 6}, 6);

    unordered_array_set res_set = unordered_array_set_complement(set,
universum);

    assert(unordered_array_set_isSubset(res_set, universum));

    unordered_array_set_delete(&set);
    unordered_array_set_delete(&universum);
    unordered_array_set_delete(&res_set);
}


void test_unordered_array_set_complement() {
    test_unordered_array_set_complement_1();
    test_unordered_array_set_complement_2();
    test_unordered_array_set_complement_3();
}


void test_unordered_array_set() {
    test_unordered_array_set_in();
    test_unordered_array_set_isSubset();
    test_unordered_array_set_deleteElement();
    test_unordered_array_set_union();
    test_unordered_array_set_intersection();
    test_unordered_array_set_difference();
    test_unordered_array_set_symmetricDifference();
    test_unordered_array_set_complement();
}




void test() {
    test_bitset();
    test_ordered_array_set();
    test_unordered_array_set();
}


int main() {
    test();

    return 0;
}
```

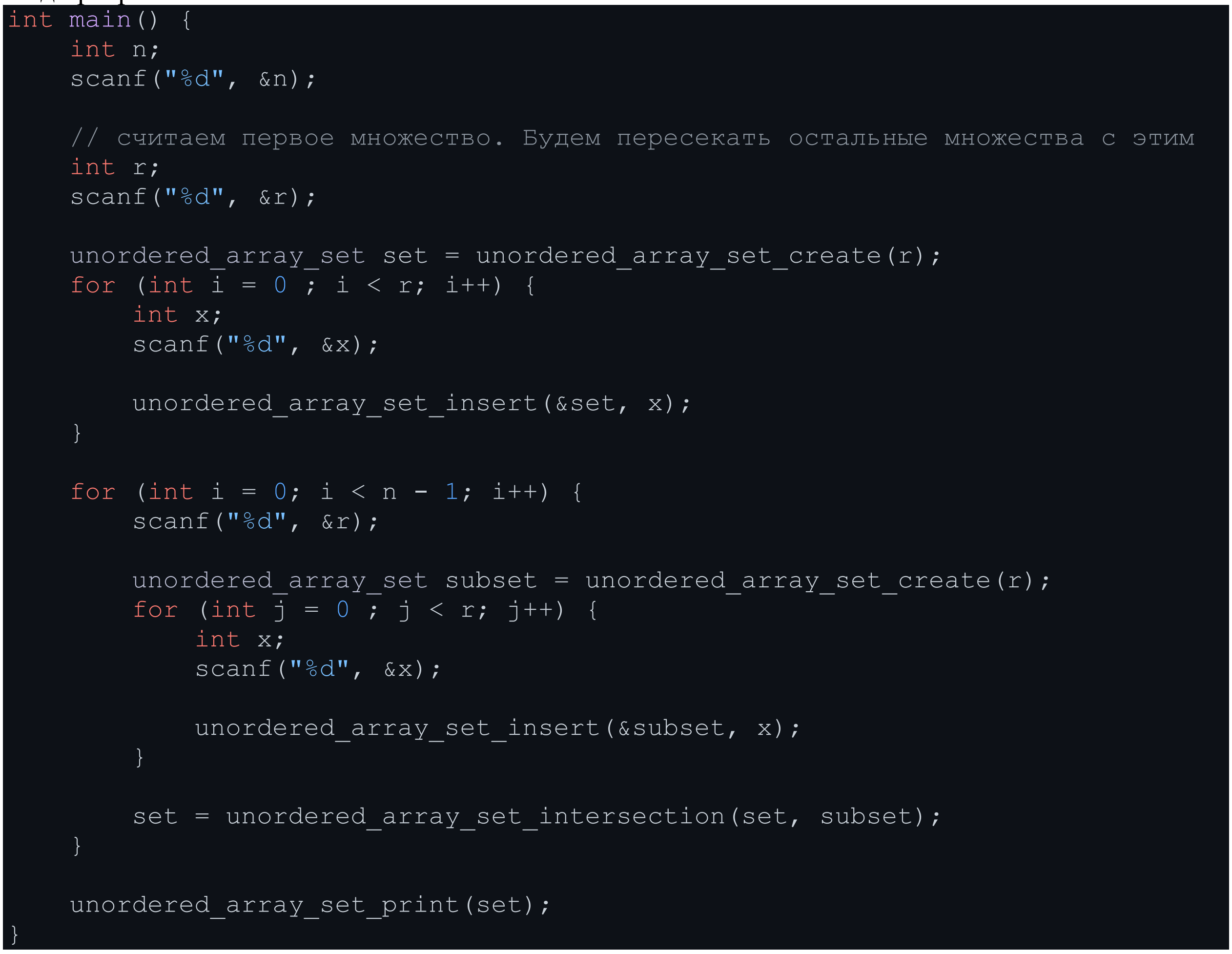
Тестовая система не вызвала ошибок, следовательно, функции работают верно:



```
D:\prjct\libary\array\cmake-build-debug\array.exe


Process finished with exit code 0
```

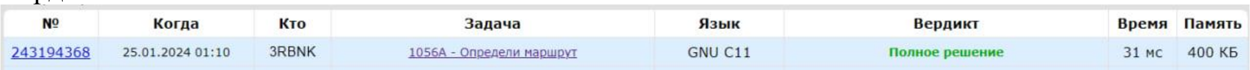
Задания с Codeforces:

Задание 1. Определи маршрут (1056A):
Код программы:



```c
int main() {
    int n;
    scanf("%d", &n);

    // считаем первое множество. Будем пересекать остальные множества с этим
    int r;
    scanf("%d", &r);

    unordered_array_set set = unordered_array_set_create(r);
    for (int i = 0 ; i < r; i++) {
        int x;
        scanf("%d", &x);

        unordered_array_set_insert(&set, x);
    }

    for (int i = 0; i < n - 1; i++) {
        scanf("%d", &r);

        unordered_array_set subset = unordered_array_set_create(r);
        for (int j = 0 ; j < r; j++) {
            int x;
            scanf("%d", &x);

            unordered_array_set_insert(&subset, x);
        }

        set = unordered_array_set_intersection(set, subset);
    }

    unordered_array_set_print(set);
}
```

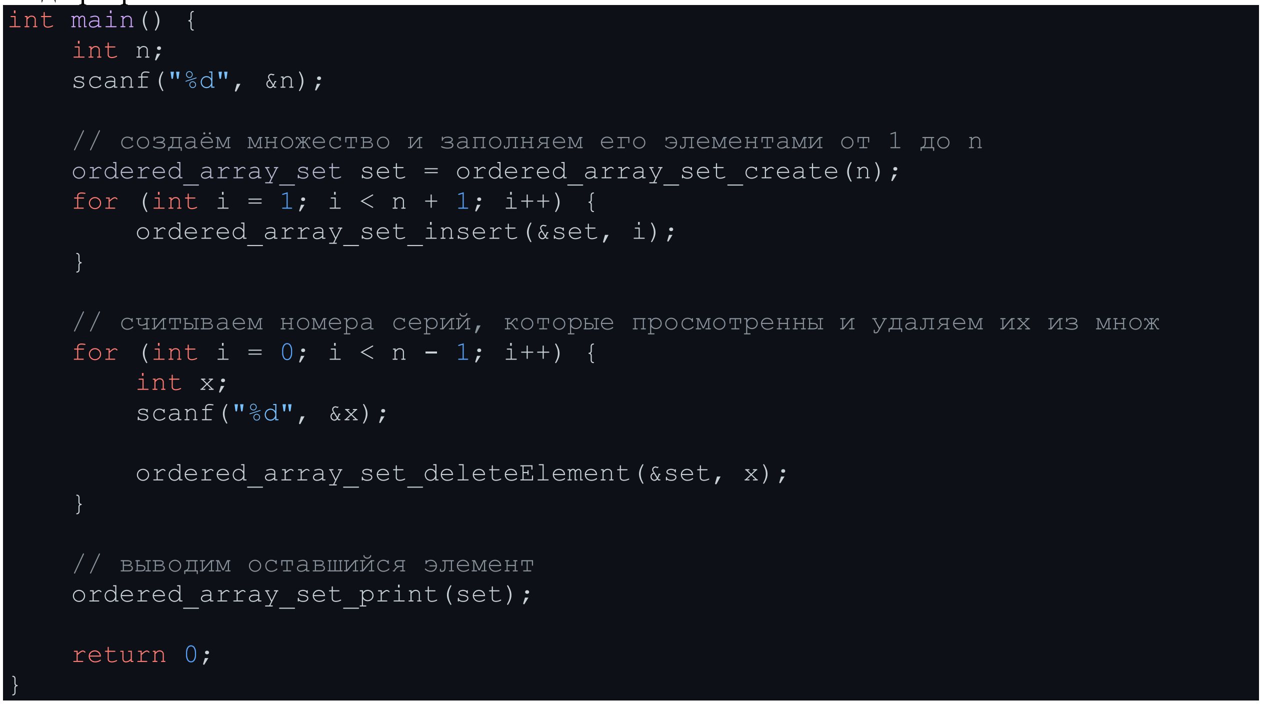
Вердикт тестовой системы:



№	Когда	Кто	Задача	Язык	Вердикт	Время	Память
243194368	25.01.2024 01:10	3RBNK	1056A - Определи маршрут	GNU C11	Полное решение	31 мс	400 КБ

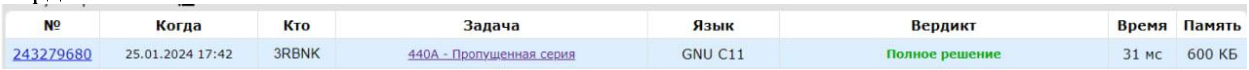
Задание 2. Пропущенная серия (440A):

Код программы:



```
int main() {
    int n;
    scanf("%d", &n);

    // создаём множество и заполняем его элементами от 1 до n
    ordered_array_set set = ordered_array_set_create(n);
    for (int i = 1; i < n + 1; i++) {
        ordered_array_set_insert(&set, i);
    }

    // считываем номера серий, которые просмотренны и удаляем их из множ
    for (int i = 0; i < n - 1; i++) {
        int x;
        scanf("%d", &x);

        ordered_array_set_deleteElement(&set, x);
    }

    // выводим оставшийся элемент
    ordered_array_set_print(set);

    return 0;
}
```

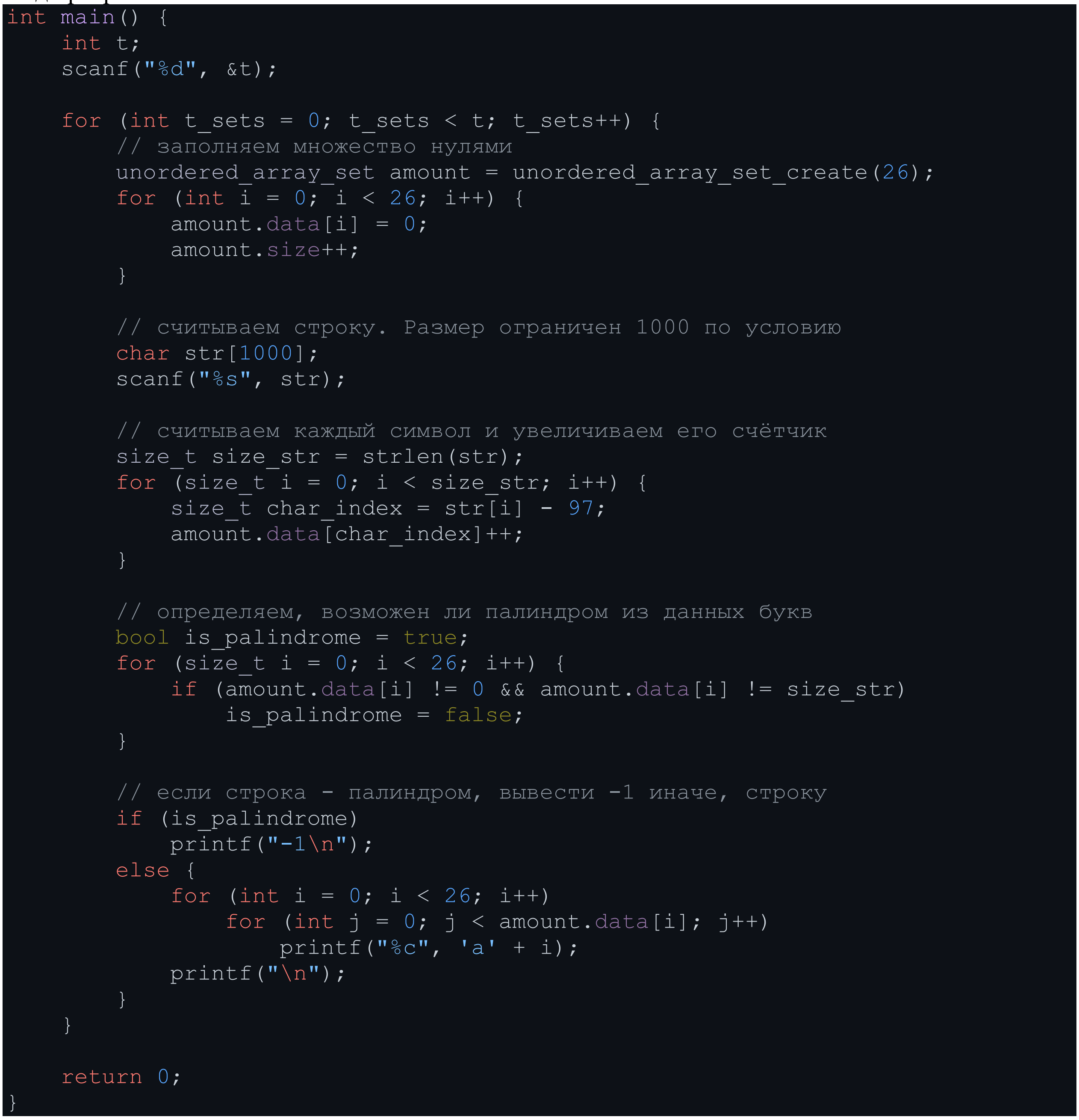
Вердикт тестовой системы:



№	Когда	Кто	Задача	Язык	Вердикт	Время	Память
243279680	25.01.2024 17:42	3RBNK	440A - Пропущенная серия	GNU C11	Полное решение	31 мс	600 КБ

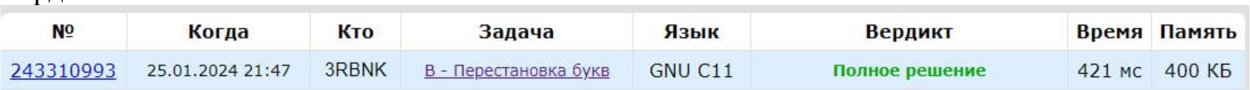
Задание 3. Перестановка букв (1093):

Код программы:



```
int main() {
    int t;
    scanf("%d", &t);

    for (int t_sets = 0; t_sets < t; t_sets++) {
        // заполняем множество нулями
        unordered_array_set amount = unordered_array_set_create(26);
        for (int i = 0; i < 26; i++) {
            amount.data[i] = 0;
            amount.size++;
        }

        // считываем строку. Размер ограничен 1000 по условию
        char str[1000];
        scanf("%s", str);

        // считываем каждый символ и увеличиваем его счётчик
        size_t size_str = strlen(str);
        for (size_t i = 0; i < size_str; i++) {
            size_t char_index = str[i] - 97;
            amount.data[char_index]++;
        }

        // определяем, возможен ли палиндром из данных букв
        bool is_palindrome = true;
        for (size_t i = 0; i < 26; i++) {
            if (amount.data[i] != 0 && amount.data[i] != size_str)
                is_palindrome = false;
        }

        // если строка - палиндром, вывести -1 иначе, строку
        if (is_palindrome)
            printf("-1\n");
        else {
            for (int i = 0; i < 26; i++)
                for (int j = 0; j < amount.data[i]; j++)
                    printf("%c", 'a' + i);
            printf("\n");
        }
    }

    return 0;
}
```

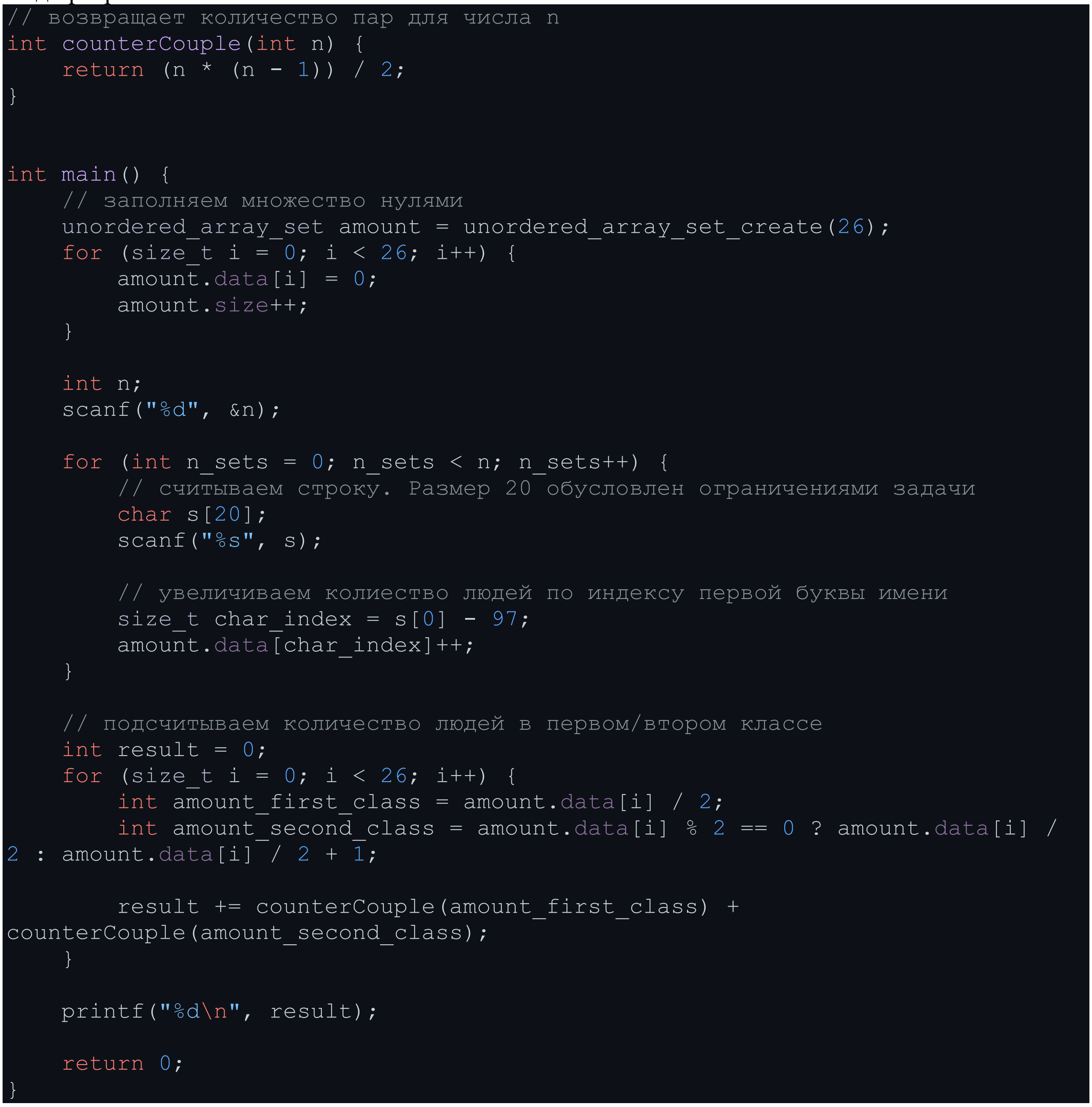
Вердикт тестовой системы:



№	Когда	Кто	Задача	Язык	Вердикт	Время	Память
243310993	25.01.2024 21:47	3RBNK	B - Перестановка букв	GNU C11	Полное решение	421 мс	400 КБ

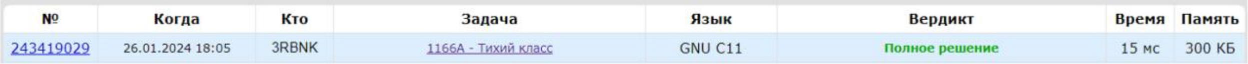
Задание 4. Тихий класс (1166A):

Код программы:



```
// возвращает количество пар для числа n
int counterCouple(int n) {
    return (n * (n - 1)) / 2;
}


int main() {
    // заполняем множество нулями
    unordered_array_set amount = unordered_array_set_create(26);
    for (size_t i = 0; i < 26; i++) {
        amount.data[i] = 0;
        amount.size++;
    }

    int n;
    scanf("%d", &n);

    for (int n_sets = 0; n_sets < n; n_sets++) {
        // считываем строку. Размер 20 обусловлен ограничениями задачи
        char s[20];
        scanf("%s", s);

        // увеличиваем колиество людей по индексу первой буквы имени
        size_t char_index = s[0] - 97;
        amount.data[char_index]++;
    }

    // подсчитываем количество людей в первом/втором классе
    int result = 0;
    for (size_t i = 0; i < 26; i++) {
        int amount_first_class = amount.data[i] / 2;
        int amount_second_class = amount.data[i] % 2 == 0 ? amount.data[i] /
2 : amount.data[i] / 2 + 1;

        result += counterCouple(amount_first_class) +
counterCouple(amount_second_class);
    }

    printf("%d\n", result);

    return 0;
}
```

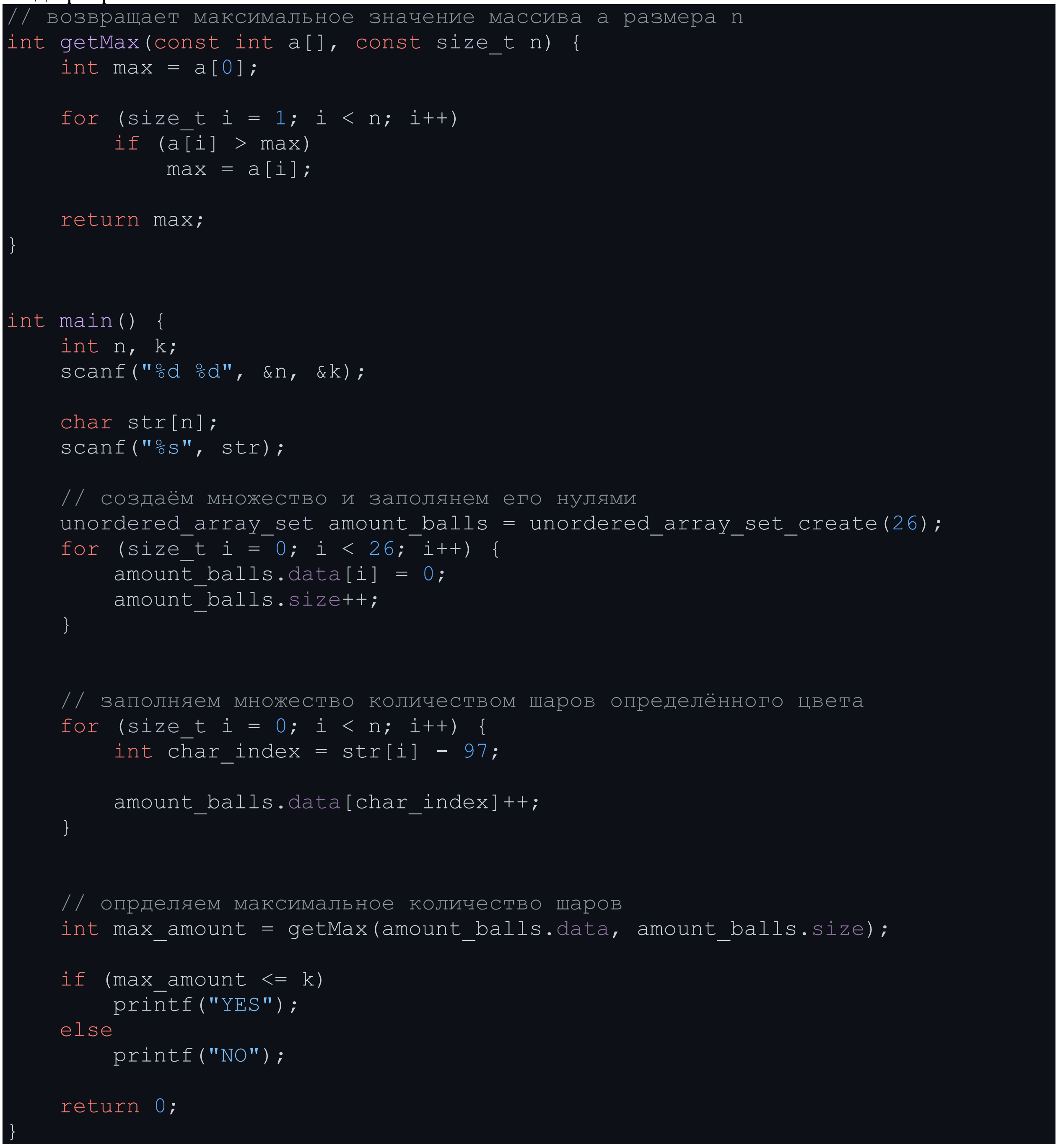
Вердикт тестовой системы:



№	Когда	Кто	Задача	Язык	Вердикт	Время	Память
243419029	26.01.2024 18:05	3RBNK	1166A - Тихий класс	GNU C11	Полное решение	15 мс	300 КБ

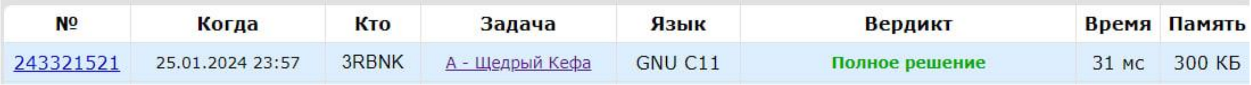
Задание 5. Щедрый Кефа (841A):

Код программы:



```
// возвращает максимальное значение массива a размера n
int getMax(const int a[], const size_t n) {
    int max = a[0];

    for (size_t i = 1; i < n; i++)
        if (a[i] > max)
            max = a[i];

    return max;
}


int main() {
    int n, k;
    scanf("%d %d", &n, &k);

    char str[n];
    scanf("%s", str);

    // создаём множество и заполянем его нулями
    unordered_array_set amount_balls = unordered_array_set_create(26);
    for (size_t i = 0; i < 26; i++) {
        amount_balls.data[i] = 0;
        amount_balls.size++;
    }


    // заполняем множество количеством шаров определённого цвета
    for (size_t i = 0; i < n; i++) {
        int char_index = str[i] - 97;

        amount_balls.data[char_index]++;
    }


    // опрделяем максимальное количество шаров
    int max_amount = getMax(amount_balls.data, amount_balls.size);

    if (max_amount <= k)
        printf("YES");
    else
        printf("NO");

    return 0;
}
```

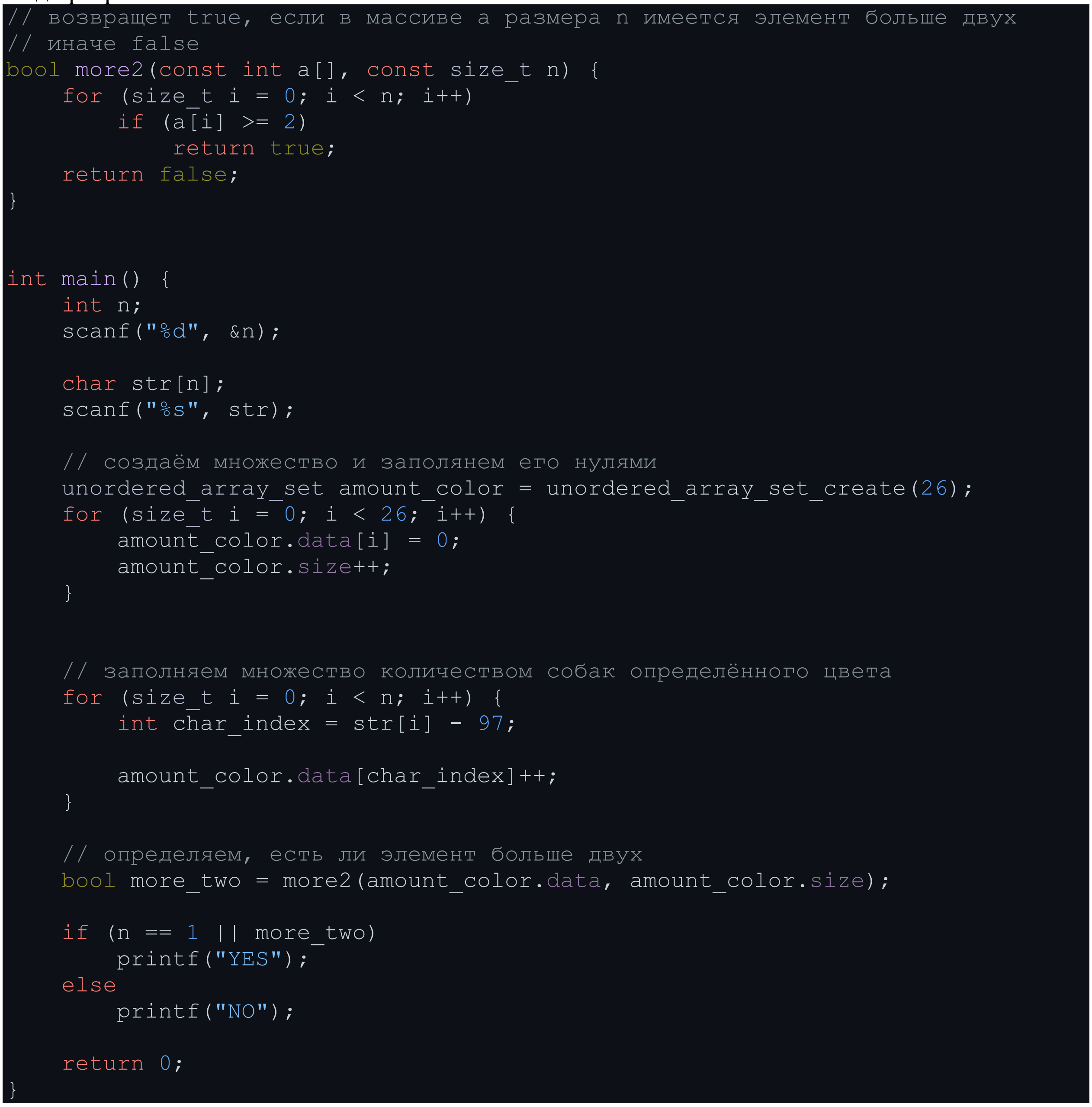
Вердикт тестовой системы:



№	Когда	Кто	Задача	Язык	Вердикт	Время	Память
243321521	25.01.2024 23:57	3RBNK	A - Щедрый Кефа	GNU C11	Полное решение	31 мс	300 КБ

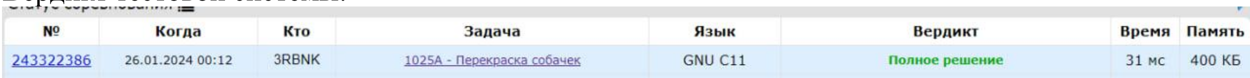
Задание 6. Перекраска собачек (1025A):

Код программы:



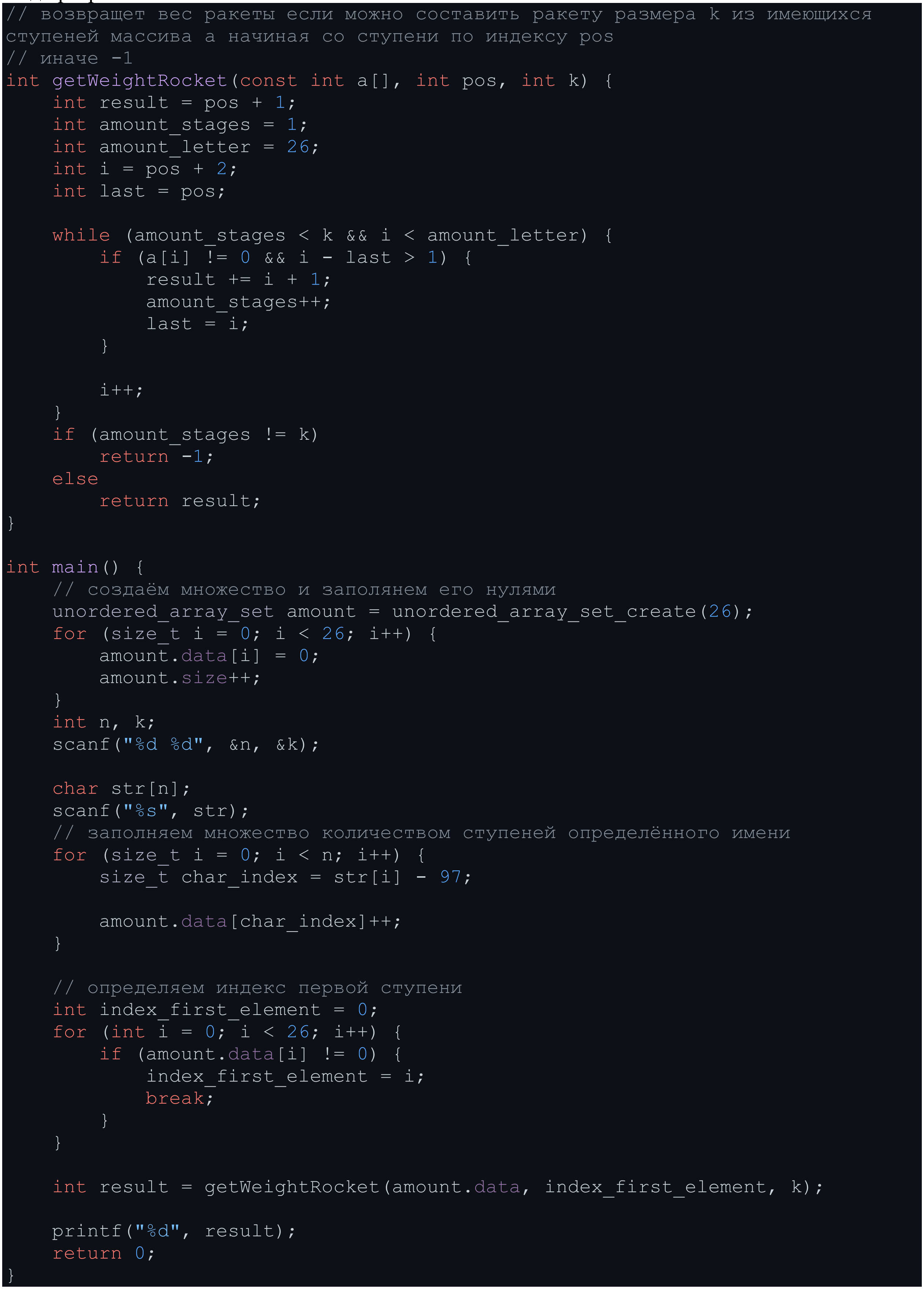
```c
// возвращет true, если в массиве a размера n имеется элемент больше двух
// иначе false
bool more2(const int a[], const size_t n) {
    for (size_t i = 0; i < n; i++)
        if (a[i] >= 2)
            return true;
    return false;
}


int main() {
    int n;
    scanf("%d", &n);

    char str[n];
    scanf("%s", str);

    // создаём множество и заполянем его нулями
    unordered_array_set amount_color = unordered_array_set_create(26);
    for (size_t i = 0; i < 26; i++) {
        amount_color.data[i] = 0;
        amount_color.size++;
    }


    // заполняем множество количеством собак определённого цвета
    for (size_t i = 0; i < n; i++) {
        int char_index = str[i] - 97;

        amount_color.data[char_index]++;
    }

    // определяем, есть ли элемент больше двух
    bool more_two = more2(amount_color.data, amount_color.size);

    if (n == 1 || more_two)
        printf("YES");
    else
        printf("NO");

    return 0;
}
```

Вердикт тестовой системы:



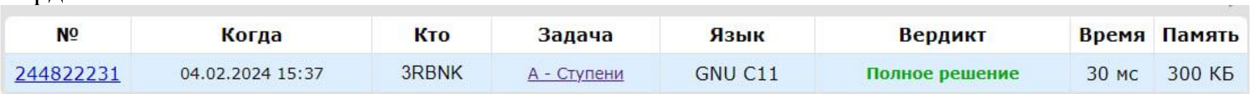
№	Когда	Кто	Задача	Язык	Вердикт	Время	Память
243322386	26.01.2024 00:12	3RBNK	1025A - Перекраска собачек	GNU C11	Полное решение	31 мс	400 КБ

Задание 7. Ступени (1011A):
Код программы:



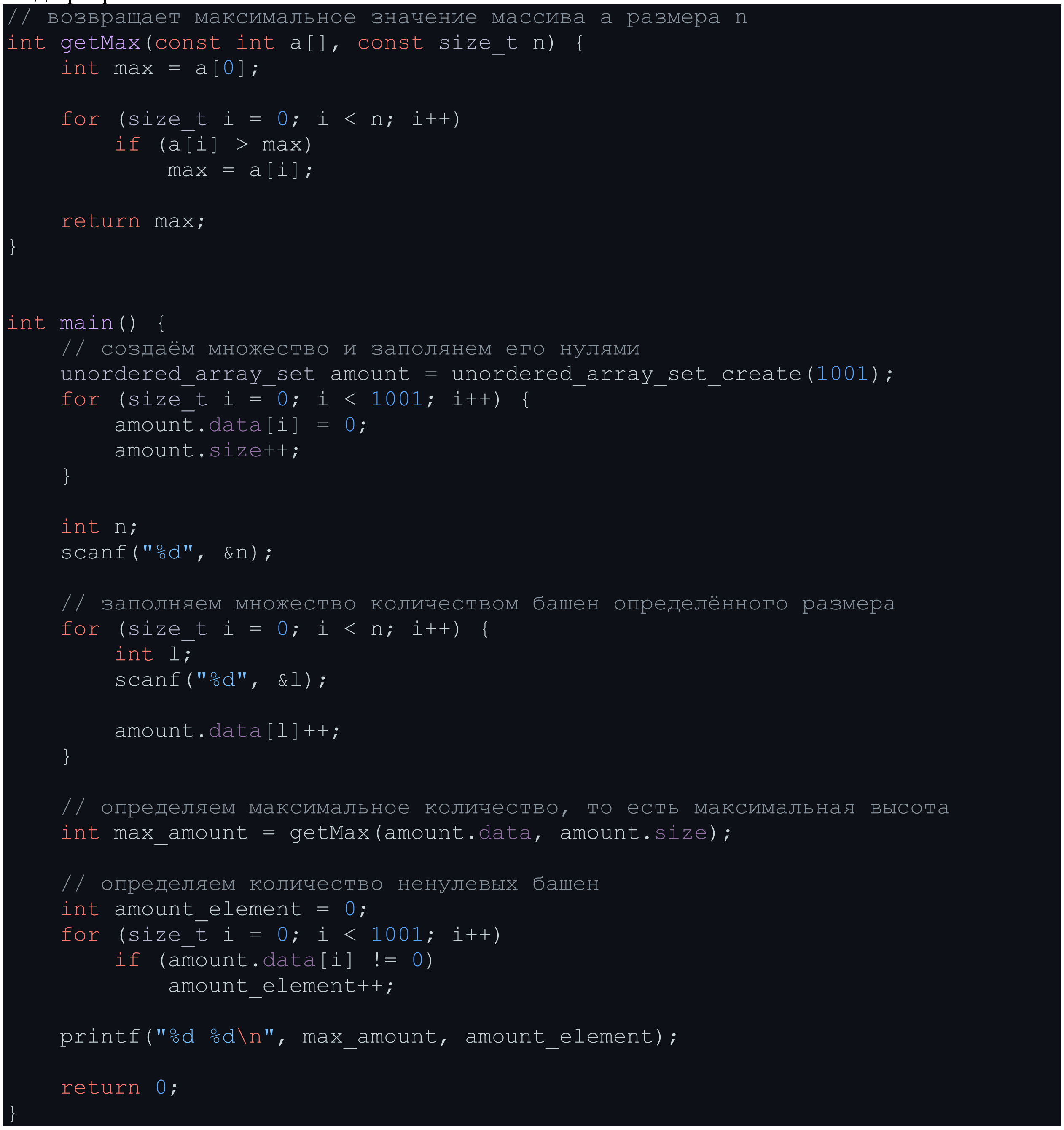
```
// возвращет вес ракеты если можно составить ракету размера k из имеющихся
ступеней массива a начиная со ступени по индексу pos
// иначе -1
int getWeightRocket(const int a[], int pos, int k) {
    int result = pos + 1;
    int amount_stages = 1;
    int amount_letter = 26;
    int i = pos + 2;
    int last = pos;

    while (amount_stages < k && i < amount_letter) {
        if (a[i] != 0 && i - last > 1) {
            result += i + 1;
            amount_stages++;
            last = i;
        }

        i++;
    }
    if (amount_stages != k)
        return -1;
    else
        return result;
}

int main() {
    // создаём множество и заполянем его нулями
    unordered_array_set amount = unordered_array_set_create(26);
    for (size_t i = 0; i < 26; i++) {
        amount.data[i] = 0;
        amount.size++;
    }
    int n, k;
    scanf("%d %d", &n, &k);

    char str[n];
    scanf("%s", str);
    // заполняем множество количеством ступеней определённого имени
    for (size_t i = 0; i < n; i++) {
        size_t char_index = str[i] - 97;

        amount.data[char_index]++;
    }

    // определяем индекс первой ступени
    int index_first_element = 0;
    for (int i = 0; i < 26; i++) {
        if (amount.data[i] != 0) {
            index_first_element = i;
            break;
        }
    }

    int result = getWeightRocket(amount.data, index_first_element, k);

    printf("%d", result);
    return 0;
}
```

Вердикт тестовой системы:



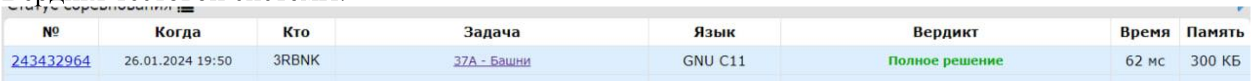
№	Когда	Кто	Задача	Язык	Вердикт	Время	Память
244822231	04.02.2024 15:37	3RBNK	A - Ступени	GNU C11	Полное решение	30 мс	300 КБ

Задание 8. Башни (37А):
Код программы:



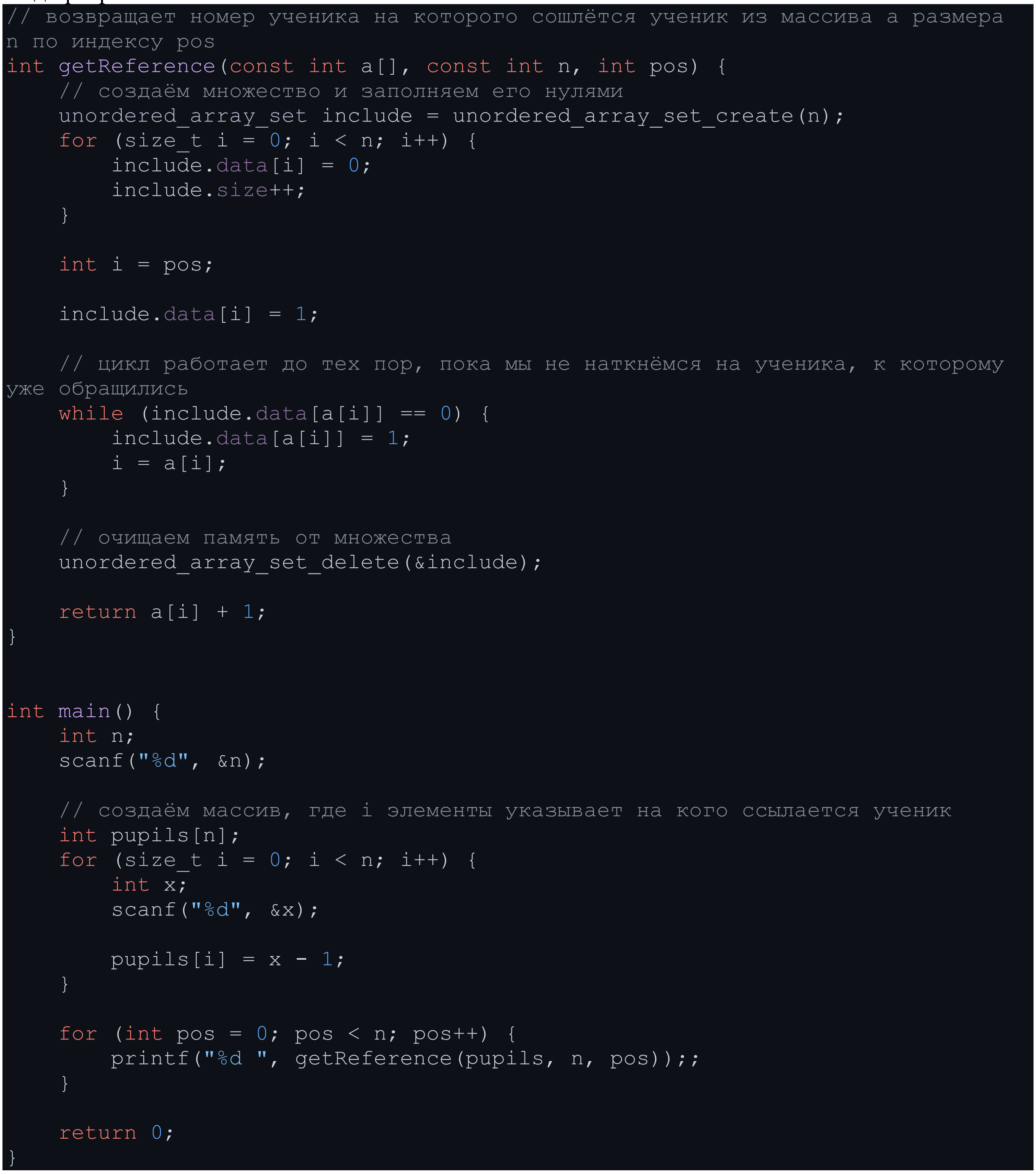
```
// возвращает максимальное значение массива a размера n
int getMax(const int a[], const size_t n) {
    int max = a[0];

    for (size_t i = 0; i < n; i++)
        if (a[i] > max)
            max = a[i];

    return max;
}


int main() {
    // создаём множество и заполянем его нулями
    unordered_array_set amount = unordered_array_set_create(1001);
    for (size_t i = 0; i < 1001; i++) {
        amount.data[i] = 0;
        amount.size++;
    }

    int n;
    scanf("%d", &n);

    // заполняем множество количеством башен определённого размера
    for (size_t i = 0; i < n; i++) {
        int l;
        scanf("%d", &l);

        amount.data[l]++;
    }

    // определяем максимальное количество, то есть максимальная высота
    int max_amount = getMax(amount.data, amount.size);

    // определяем количество ненулевых башен
    int amount_element = 0;
    for (size_t i = 0; i < 1001; i++)
        if (amount.data[i] != 0)
            amount_element++;

    printf("%d %d\n", max_amount, amount_element);

    return 0;
}
```

Вердикт тестовой системы:



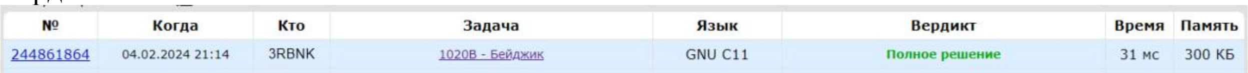
№	Когда	Кто	Задача	Язык	Вердикт	Время	Память
243432964	26.01.2024 19:50	3RBNK	37A - Башни	GNU C11	Полное решение	62 мс	300 КБ

Задание 9. Бейджик (1020B):
Код программы:



```
// возвращает номер ученика на которого сошлётся ученик из массива a размера
n по индексу pos
int getReference(const int a[], const int n, int pos) {
    // создаём множество и заполняем его нулями
    unordered_array_set include = unordered_array_set_create(n);
    for (size_t i = 0; i < n; i++) {
        include.data[i] = 0;
        include.size++;
    }

    int i = pos;

    include.data[i] = 1;

    // цикл работает до тех пор, пока мы не наткнёмся на ученика, к которому
уже обращились
    while (include.data[a[i]] == 0) {
        include.data[a[i]] = 1;
        i = a[i];
    }

    // очищаем память от множества
    unordered_array_set_delete(&include);

    return a[i] + 1;
}


int main() {
    int n;
    scanf("%d", &n);

    // создаём массив, где i элементы указывает на кого ссылается ученик
    int pupils[n];
    for (size_t i = 0; i < n; i++) {
        int x;
        scanf("%d", &x);

        pupils[i] = x - 1;
    }

    for (int pos = 0; pos < n; pos++) {
        printf("%d ", getReference(pupils, n, pos));;
    }

    return 0;
}
```

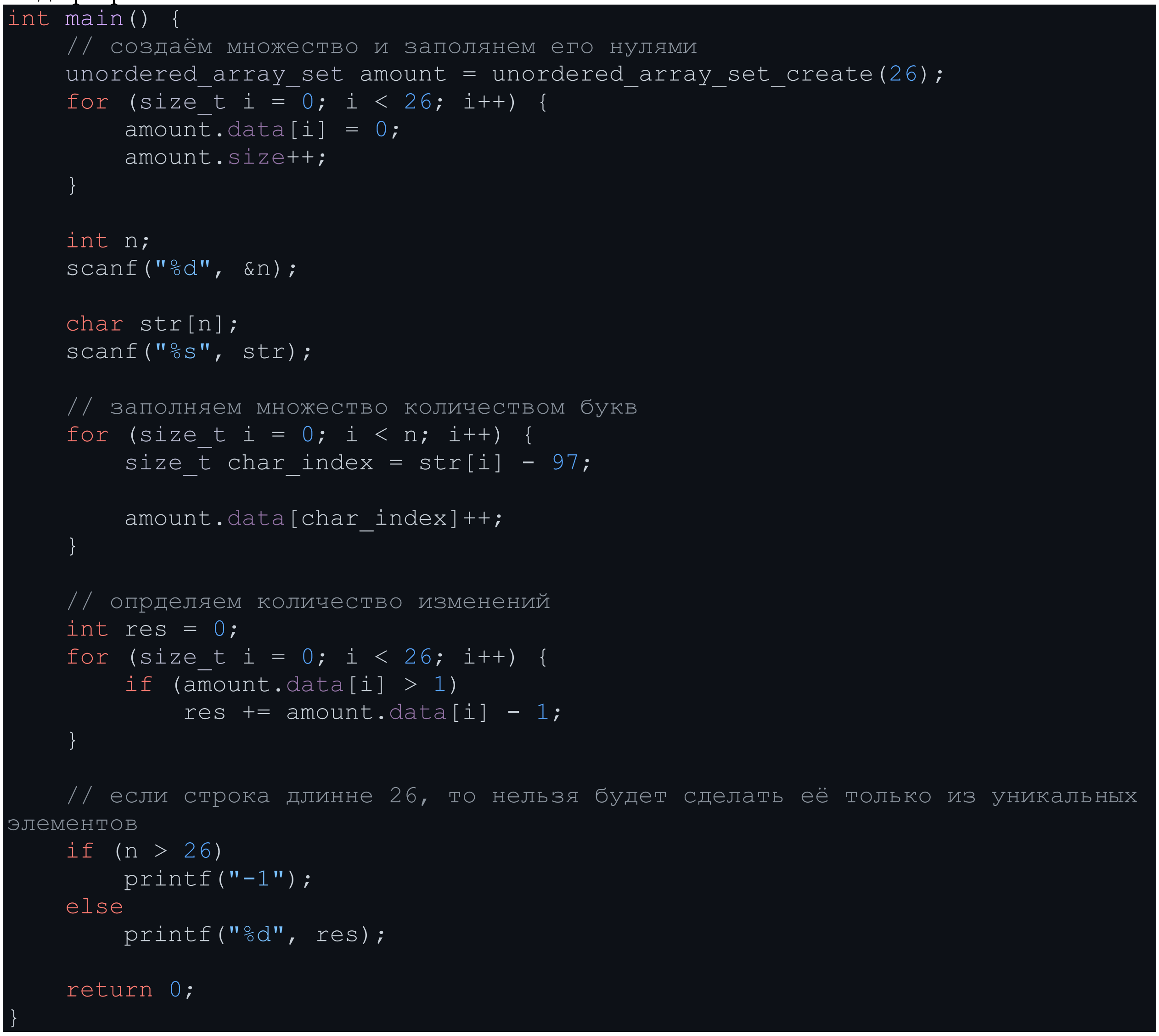
Вердикт тестовой системы:



№	Когда	Кто	Задача	Язык	Вердикт	Время	Память
244861864	04.02.2024 21:14	3RBNK	1020B - Бейджик	GNU C11	Полное решение	31 мс	300 КБ

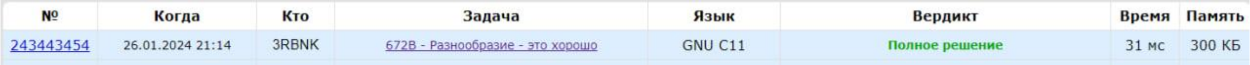
Задание 10. Разнообразие — это хорошо (672B):

Код программы:



```
int main() {
    // создаём множество и заполянем его нулями
    unordered_array_set amount = unordered_array_set_create(26);
    for (size_t i = 0; i < 26; i++) {
        amount.data[i] = 0;
        amount.size++;
    }

    int n;
    scanf("%d", &n);

    char str[n];
    scanf("%s", str);

    // заполняем множество количеством букв
    for (size_t i = 0; i < n; i++) {
        size_t char_index = str[i] - 97;

        amount.data[char_index]++;
    }

    // опрделяем количество изменений
    int res = 0;
    for (size_t i = 0; i < 26; i++) {
        if (amount.data[i] > 1)
            res += amount.data[i] - 1;
    }

    // если строка длинне 26, то нельзя будет сделать её только из уникальных элементов
    if (n > 26)
        printf("-1");
    else
        printf("%d", res);

    return 0;
}
```

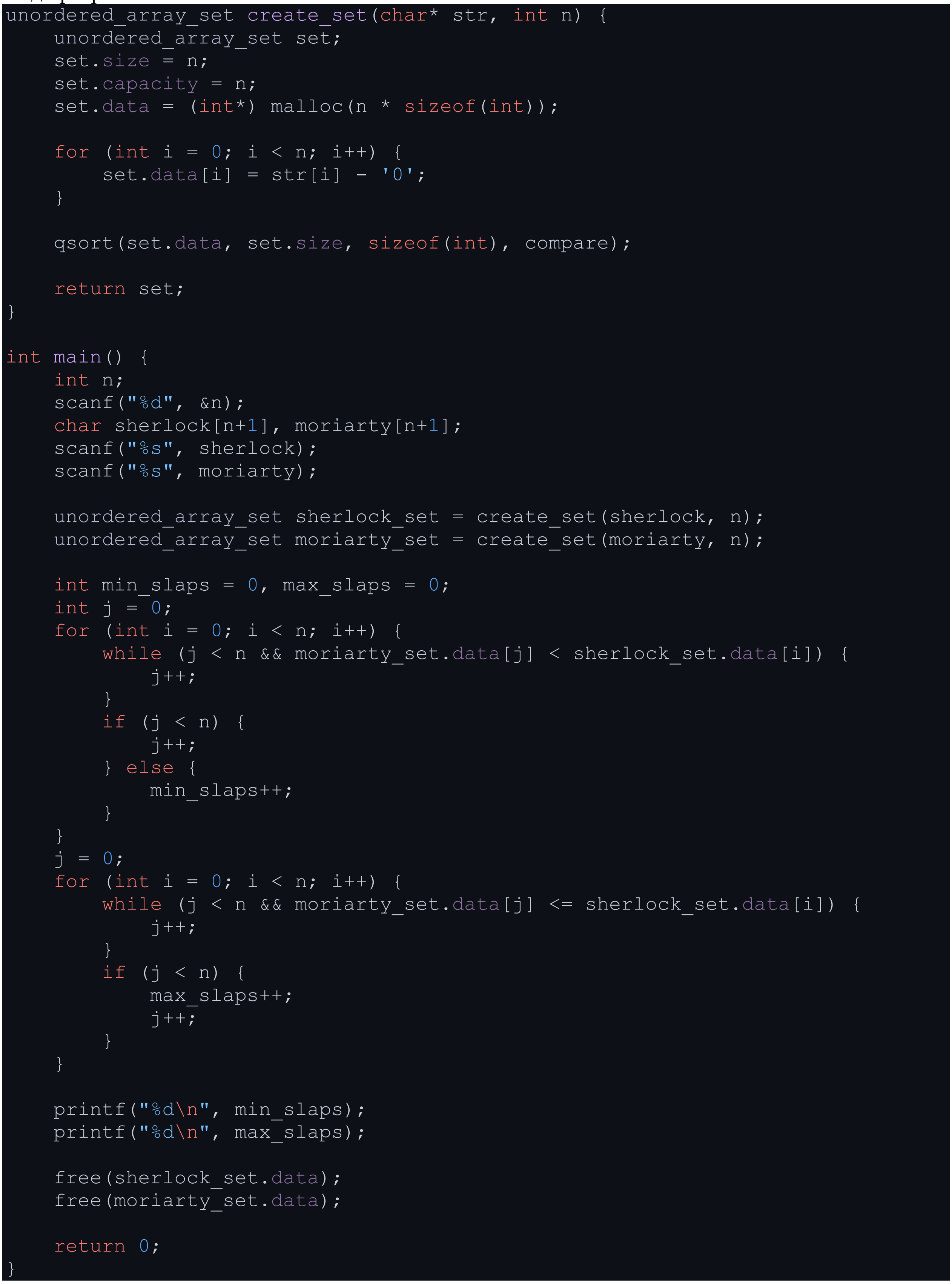
Вердикт тестовой системы:



№	Когда	Кто	Задача	Язык	Вердикт	Время	Память
243443454	26.01.2024 21:14	3RBNK	672B - Разнообразие - это хорошо	GNU C11	Полное решение	31 мс	300 КБ

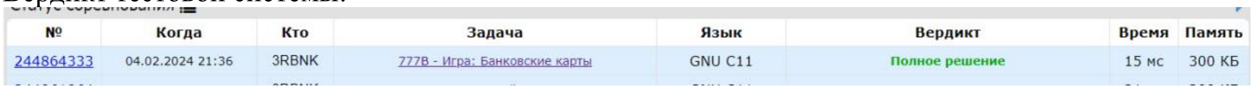
Задание 11. Игра: Банковские карты (777B):

Код программы:



```c
unordered_array_set create_set(char* str, int n) {
    unordered_array_set set;
    set.size = n;
    set.capacity = n;
    set.data = (int*) malloc(n * sizeof(int));

    for (int i = 0; i < n; i++) {
        set.data[i] = str[i] - '0';
    }

    qsort(set.data, set.size, sizeof(int), compare);

    return set;
}

int main() {
    int n;
    scanf("%d", &n);
    char sherlock[n+1], moriarty[n+1];
    scanf("%s", sherlock);
    scanf("%s", moriarty);

    unordered_array_set sherlock_set = create_set(sherlock, n);
    unordered_array_set moriarty_set = create_set(moriarty, n);

    int min_slaps = 0, max_slaps = 0;
    int j = 0;
    for (int i = 0; i < n; i++) {
        while (j < n && moriarty_set.data[j] < sherlock_set.data[i]) {
            j++;
        }
        if (j < n) {
            j++;
        } else {
            min_slaps++;
        }
    }
    j = 0;
    for (int i = 0; i < n; i++) {
        while (j < n && moriarty_set.data[j] <= sherlock_set.data[i]) {
            j++;
        }
        if (j < n) {
            max_slaps++;
            j++;
        }
    }

    printf("%d\n", min_slaps);
    printf("%d\n", max_slaps);

    free(sherlock_set.data);
    free(moriarty_set.data);

    return 0;
}
```

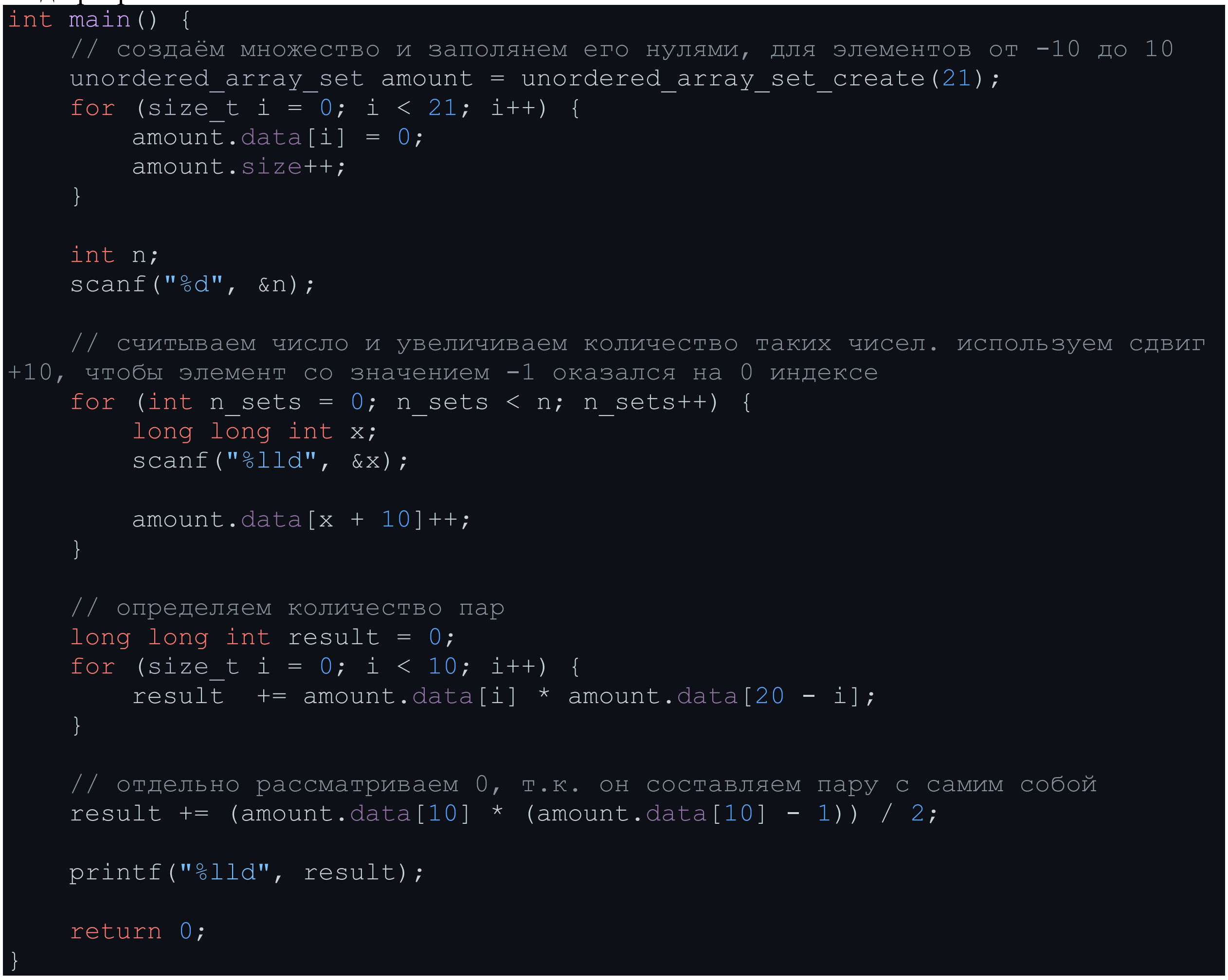
Вердикт тестовой системы:



№	Когда	Кто	Задача	Язык	Вердикт	Время	Память
244864333	04.02.2024 21:36	3RBNK	777B - Игра: Банковские карты	GNU C11	Полное решение	15 мс	300 КБ

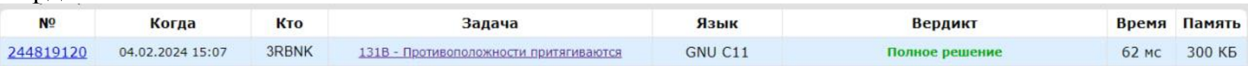
Задание 12. Противоположности притягиваются (131B):

Код программы:



```
int main() {
    // создаём множество и заполянем его нулями, для элементов от -10 до 10
    unordered_array_set amount = unordered_array_set_create(21);
    for (size_t i = 0; i < 21; i++) {
        amount.data[i] = 0;
        amount.size++;
    }

    int n;
    scanf("%d", &n);

    // считываем число и увеличиваем количество таких чисел. используем сдвиг
+10, чтобы элемент со значением -1 оказался на 0 индексе
    for (int n_sets = 0; n_sets < n; n_sets++) {
        long long int x;
        scanf("%lld", &x);

        amount.data[x + 10]++;
    }

    // определяем количество пар
    long long int result = 0;
    for (size_t i = 0; i < 10; i++) {
        result  += amount.data[i] * amount.data[20 - i];
    }

    // отдельно рассматриваем 0, т.к. он составляем пару с самим собой
    result += (amount.data[10] * (amount.data[10] - 1)) / 2;

    printf("%lld", result);

    return 0;
}
```

Вердикт тестовой системы:



№	Когда	Кто	Задача	Язык	Вердикт	Время	Память
244819120	04.02.2024 15:07	3RBNK	131B - Противоположности притягиваются	GNU C11	Полное решение	62 мс	300 КБ

Вывод: в ходе выполнения лабораторной работы мы успешно закрепили навыки работы с структурами данных и изучили простые способы представления множеств в памяти ЭВМ. Работа с структурами позволяет нам организовывать данные таким образом, чтобы обеспечить удобство и эффективность их обработки. Изучение способов представления множеств в памяти ЭВМ дало нам понимание того, как компьютер хранит и обрабатывает информацию о множествах, что является важным аспектом в разработке программного обеспечения. Эта лабораторная работа помогла нам не только углубить знания о структурах данных, но и понять, как они используются на практике для решения конкретных задач. Мы убедились в важности эффективного управления данными и их представлении для создания эффективных и мощных программ. Благодаря этой работе мы теперь осознаем, что понимание работы со структурами данных и способами представления множеств помогает нам стать более компетентными программистами и решать задачи более эффективно.